**Туккель Н.И., Шалыто А.А., Верба М.Т.**

**(aurora@peterlink.ru)**

# ПРИМЕР ПРОГРАММНОЙ ДОКУМЕНТАЦИИ НА ПОДСИСТЕМУ УПРАВЛЕНИЯ ПЕЧАТЬЮ, РАЗРАБОТАННЫЙ НА ОСНОВЕ SWITCH-ТЕХНОЛОГИИ

Санкт-Петербург
2001

# СОДЕРЖАНИЕ

# 1. ВВЕДЕНИЕ

Предлагаемый пример проектной документации реально разработанного и внедренного программного обеспечения демонстрирует результат применения SWITCH-технологии, описанной в статье Шалыто А.А., Туккель Н.И. "SWITCH-технология — автоматный подход к созданию программного обеспечения «реактивных» систем" (www.softcraft.ru).

Рассматриваемое программное обеспечение разработано на основе выданного Заказчиком технического задания, которое также приведено ниже с незначительными изменениями.

Разработанная подсистема состоит из двух частей:

- менеджера печати, функционирующего как отдельный процесс (автомат выполнения печати);
- модуля контроля режима печати, внедряемого в использующее его программное обеспечение в виде библиотеки (автомат контроля режима печати).

Менеджер печати и модуль контроля режима печати взаимодействуют асинхронно через два файла, расположенные на RAM-диске.

Фрагмент системозависимой части модуля контроля печати также реализован с помощью автомата (автомат буферизации предварительного просмотра).

Отметим, что рассмотренный пример является достаточно "простым" в рамках предложенной технологии, так как в нем автоматы не взаимодействуют друг с другом ни по одному из трех указанных в статье способов (вложенность, вызываемость, обмен номерами состояний).

# 2. ЧАСТНОЕ ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА ПОДСИСТЕМУ УПРАВЛЕНИЯ ПЕЧАТЬЮ

## 1. ВВЕДЕНИЕ

Наименование разработки: “Программное обеспечение подсистемы управления печатью” (далее "подсистема").

Подсистема управления печатью является частью программного обеспечения разрабатываемой системы.

## 2. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Подсистема предназначена для реализации различных режимов печати протоколов функционирования системы с возможностью их предварительного просмотра в среде графической оболочки Photon.

## 3. ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

### 3.1.Требования к функциональным характеристикам

3.1.1.Требования к инструментальной среде

3.1.1.1.Подсистема предназначена для функционирования под управлением ОС QNX версии 4.25 и графической оболочки Photon версии 1.14.

3.1.1.2.Разработка подсистемы выполняется с использованием компилятора Watcom C версии 10.6 и инструментальных средств Photon Application Builder версии 1.14.

3.1.2.Требования к пользовательскому интерфейсу

3.1.2.1.Подсистема работает как отдельный процесс и может запускаться из прикладной задачи.

3.1.2.2.Подсистема функционирует в графическом разрешении не ниже, чем 1024x768 пикселей.

3.1.2.3.Размер окна подсистемы составляет 800x480 пикселей.

3.1.2.4.Во включенном (нажатом) положении кнопки должны быть закрашены зеленым цветом. Отключенному (не нажатому) положению кнопок соответствует их закраска белым цветом. В заблокированном состоянии кнопок надписи на кнопках должны быть серого цвета, а анимация нажатия кнопок должна быть отключена.

3.1.2.5.Команды на управление поступают в соответствии с режимом печати (печать файла или строки).

3.1.3.Требования к реализуемым функциям

3.1.3.1.После запуска подсистема может находится в одном из трех режимов печати: “Печать по вызову”, “Печать по времени” и “Автоматическая печать”.

3.1.3.2.В режиме “Печать по вызову” оператор может выбрать для печати любой из имеющихся в указанном каталоге текстовых файлов. После выбора файла его предварительный просмотр и печать выполняются после нажатия оператором соответствующих кнопок.

3.1.3.3.В режиме “Печать по времени” оператор может задать интервал времени, через который будут автоматически печататься

выбранные файлы. Выбор файлов осуществляется из статического списка. Отсчет временного интервала начинается в момент задания его значения. Проверка, какие именно файлы выбраны для печати, осуществляется в момент истечения заданного интервала времени. Кроме этого, оператор может осуществлять предварительный просмотр и печать выбранных файлов, не дожидаясь окончания заданного временного интервала, по аналогии с тем, как это выполняется в режиме “Печать по вызову”.

3.1.3.4.В режиме “Автоматическая печать” осуществляется предварительный просмотр и печать текстовых строк, поступающих от функциональных программ (ФП). При переключении в этот режим автоматически выполняется печать фиксировано заданного файла (заголовка). Кнопки предварительного просмотра и печати в этом режиме должны быть заблокированы. В случае достижения временным файлом печати максимально допустимого объема, все поступающие от ФП запросы на печать должны игнорироваться, пока содержимое временного файла печати не будет допечатано до конца, после чего файл урезается до нулевой длины.

3.1.3.5.Должна быть предусмотрена возможность приостановки печати по команде оператора.

3.1.3.6.Должна быть предусмотрена возможность прекращения печати по команде оператора с очисткой очереди печати.

3.1.3.7.Должно быть реализовано корректное отображение символов кириллицы и символов псевдографики в окне предварительного просмотра.

3.1.3.8.Распечатка каждого файла должна предваряться строкой, содержащей дату и время отправки данного файла на печать.

3.1.3.9.Должна быть предусмотрена возможность разбивки на страницы с разделительной линией ("-" или "=") и номером листа (общего количества листов), а также установкой колонтитула (имя файла или номер протокола).

## 3.2.Требования к надежности

Должна быть предусмотрена защита от некорректного одновременного доступа к печатаемому файлу.

## 3.3.Требования к составу и параметрам технических средств

3.3.1.Подсистема предназначена для функционирования на крейте VC-7.

3.3.2.В качестве печатающего устройства должен использоваться подключенный к последовательному порту принтер УД-М312. Ширина строки принимается равной 78 символам. Размер страницы 50 строк.

3.3.3.Должна быть реализована возможность ограничения максимального размера временных файлов, создаваемых при работе подсистемы.

3.3.4.Особые требования к используемому объему оперативной памяти и производительности не предъявляются.

## 3.4.Требования к информационной и программной совместимости

3.4.1.Обмен информацией между ФП и подсистемой осуществляется с использованием асинхронных механизмов. Подсистема получает от ФП текстовую строку, которая обрезается до принятой для печати длины строки (см. п. 3.3.2) и обрабатывается в зависимости от состояния

подсистемы. В ответном сообщении подсистема возвращает результат обработки сообщения, кодируемый следующим образом:
0 - сообщение добавлено в очередь печати;
1 - сообщение проигнорировано, так как подсистема находится не в режиме “Автоматическая печать”;
2 - сообщение проигнорировано, так как временный файл печати переполнился.

### 3.5.Специальные требования

Специальные требования не предъявляются.

## 4. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

По окончанию проектирования Разработчик предоставляет Заказчику следующие документы:
1) Перечни событий, входных переменных и выходных воздействий, участвующих в работе подсистемы.
2) Схемы связей и графы переходов конечных автоматов, специфицирующих поведение подсистемы.
3) Протоколы работы подсистемы для различных режимов работы.

## 5. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Требования к технико-экономическим показателям не предъявляются.

## 6. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

1) Разработка и отладка автономной версии подсистемы.
2) Интеграция подсистемы в систему.

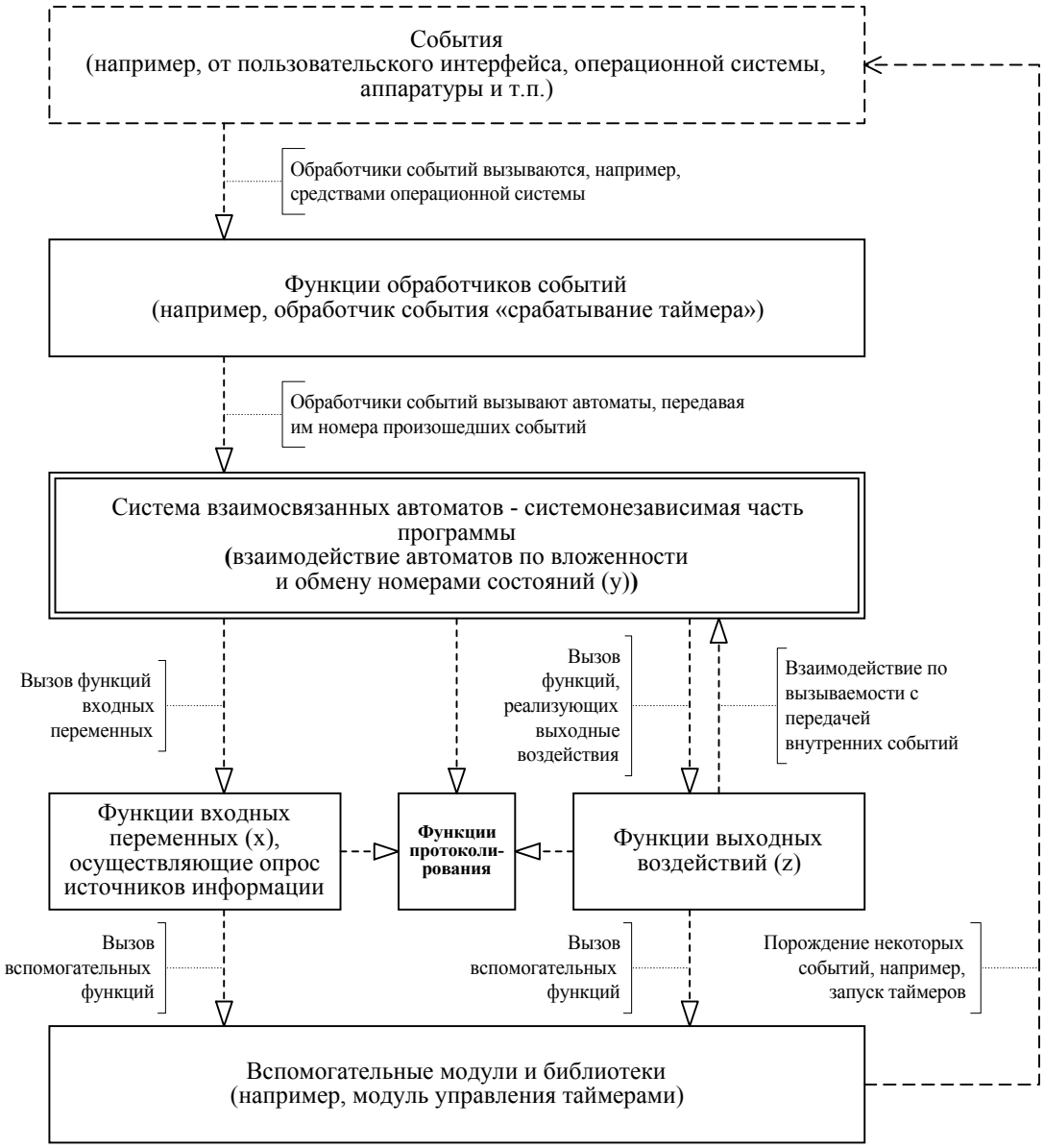
## 7. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

Приемка подсистемы осуществляется в три этапа:
1) Проверка функционирования автономной версии подсистемы на инструментальной ЭВМ с имитацией распечатки контрольных примеров, составленных Разработчиком.
2) Проверка функционирования автономной версии подсистемы на крейте VC-7 с распечаткой контрольных примеров, предоставляемых Заказчиком, на принтере УД-М312.
3) Комплексная проверка функционирования подсистемы в составе системы на крейте VC-7 с распечаткой контрольных примеров, предоставляемых Заказчиком, на принтере УД-М312.

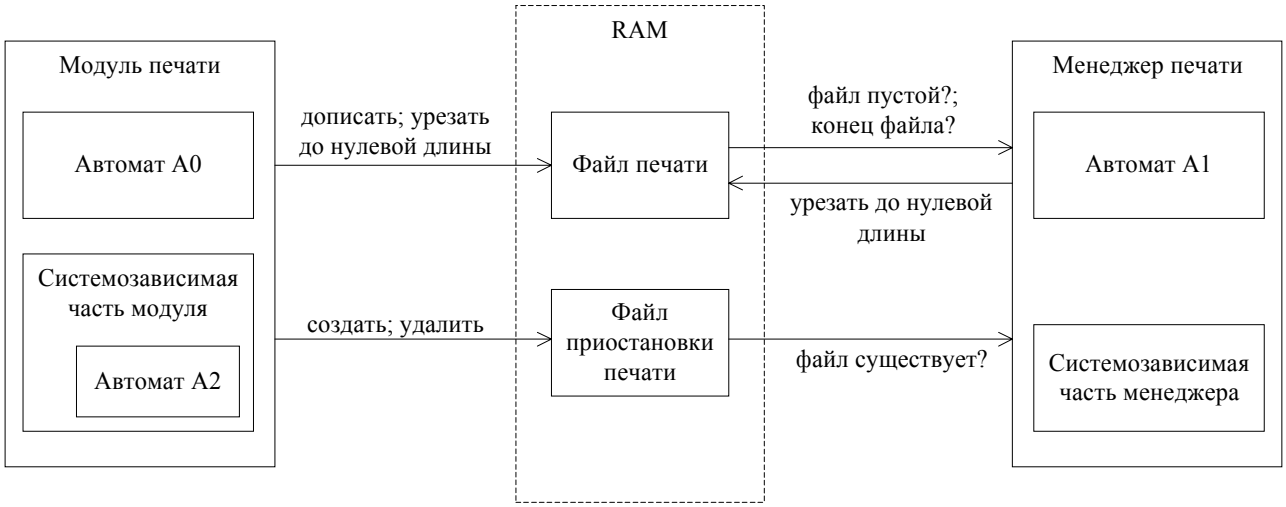
После приемки подсистемы ее дальнейшее сопровождение осуществляется Заказчиком.

# 3. СТРУКТУРНАЯ СХЕМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 3.1. Общая структурная схема программ, разрабатываемых с использованием SWITCH-технологии



## 3.2. Порядок взаимодействия частей подсистемы



# 4. ПЕРЕЧЕНЬ И НУМЕРАЦИЯ СОБЫТИЙ

| e | Описание |
|---|---|
| | **Для автомата контроля режима печати A0** |
| 11 | Выход из предварительного просмотра отдельного протокола |
| 12 | Переход из ручного режима работы в автоматический |
| 13 | Переход из автоматического режима работы в ручной |
| 30 | Нажатие кнопки ПЕЧАТЬ |
| 40 | Запуск предварительного просмотра |
| 80 | Срабатывание таймера печати по времени T80 |
| 110 | Поступление сообщения с информацией о событии |
| | **Для автомата выполнения печати A1** |
| 70 | Срабатывание таймера печати T70 |
| | **Для автомата буферизации предварительного просмотра A2** |
| 0 | Обработать очередной символ |
| 500 | Отобразить содержимое буфера |

# 5. ПЕРЕЧЕНЬ И НУМЕРАЦИЯ ВХОДНЫХ ПЕРЕМЕННЫХ

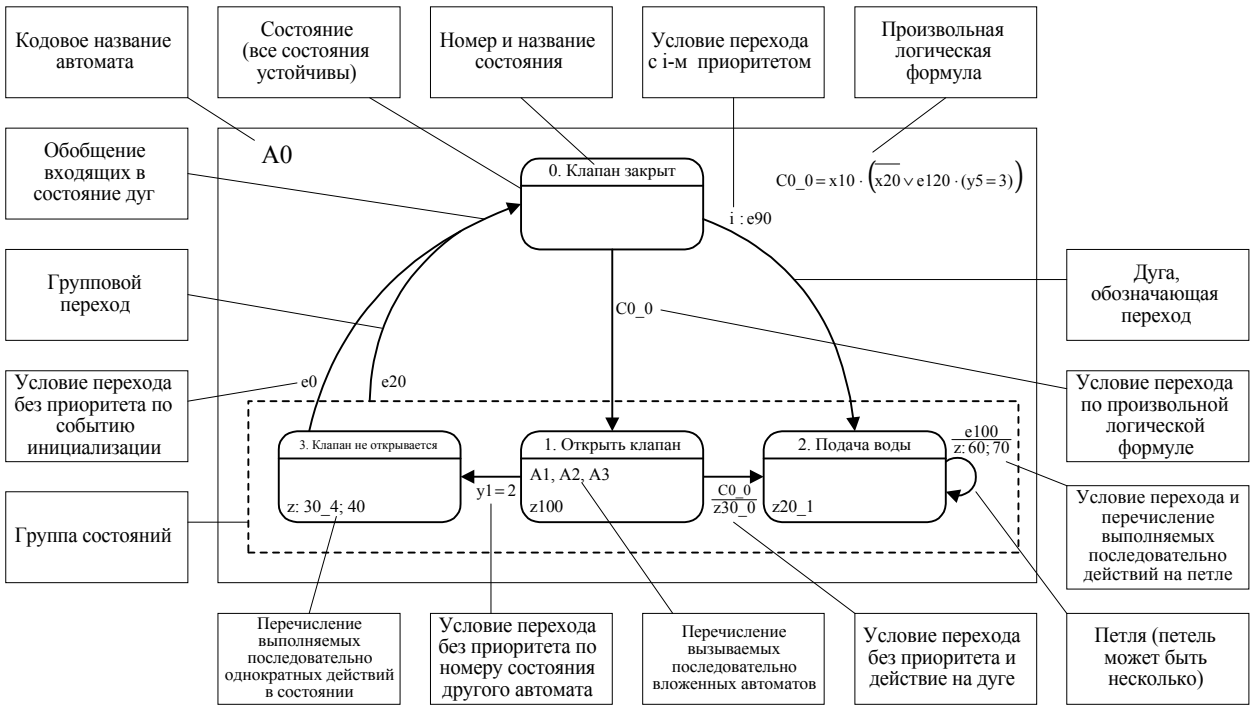
| x | Описание |
|---|---|
| | **Для автомата контроля режима печати A0** |
| 10 | Выбранный для печати по вызову файл успешно открыт |
| 70 | Размер файла печати больше допустимого |
| | **Для автомата выполнения печати A1** |
| 10 | Файл печати имеет нулевую длину |
| 50 | Конец файла печати |
| 60 | Приостановить печать |
| 100 | Порт свободен (буфер порта не переполнен) |
| | **Для автомата буферизации предварительного просмотра A2** |
| 600 | Псевдографический символ ‘–’ |
| 601 | Любой псевдографический символ |
| 602 | Символ конца строки |
| 603 | Символ пробела |

# 6. ПЕРЕЧЕНЬ И НУМЕРАЦИЯ ВЫХОДНЫХ ВОЗДЕЙСТВИЙ

| z | Описание |
|---|---|
| | **Для автомата контроля режима печати A0** |
| 45 | Очистить область предварительного просмотра |
| 50 | Урезать файл печати до нулевой длины |
| 51 | Урезать файл предварительного просмотра до нулевой длины |
| 55 | Режим приостановки печати (0 - отключить, 1 - включить/отключить) |
| 200 | Добавить выбранный файл в очередь печати |
| 210 | Добавить выбранные протоколы в очередь печати |
| 220 | Начать просмотр выбранного файла |
| 250 | Добавить в очередь печати строку с информацией о событии (автоматическая печать) |
| 260 | Добавить в очередь печати заголовок таблицы событий (автоматическая печать) |
| 500 | Возвращаемое значение (0 - сообщение обработано успешно, 1 - не включен режим автоматической печати, 2 - файл печати переполнен) |
| 800 | Показать диалог предварительного просмотра |
| | **Для автомата выполнения печати A1** |
| 50 | Урезать файл печати до нулевой длины |
| 60 | Напечатать очередной фрагмент файла печати |
| 80 | Файл печати (0 - закрыть, 1 - открыть) |
| | **Для автомата буферизации предварительного просмотра A2** |
| 600 | Отрисовать содержимое буфера |
| 601 | Буфер предварительного просмотра (0 - очистить буфер, добавить очередной символ на первую позицию и запомнить номер текущего столбца, 1 - добавить очередной символ в буфер) |
| 610 | Отрисовать псевдографический символ |
| 620 | Нарисовать горизонтальную линию |
| 621 | Значение длины горизонтальной линии (0 - присвоить значение 1, 1 - увеличить на 1) |

# 7. ПОЯСНЕНИЯ К ИСПОЛЬЗУЕМОЙ НОТАЦИИ

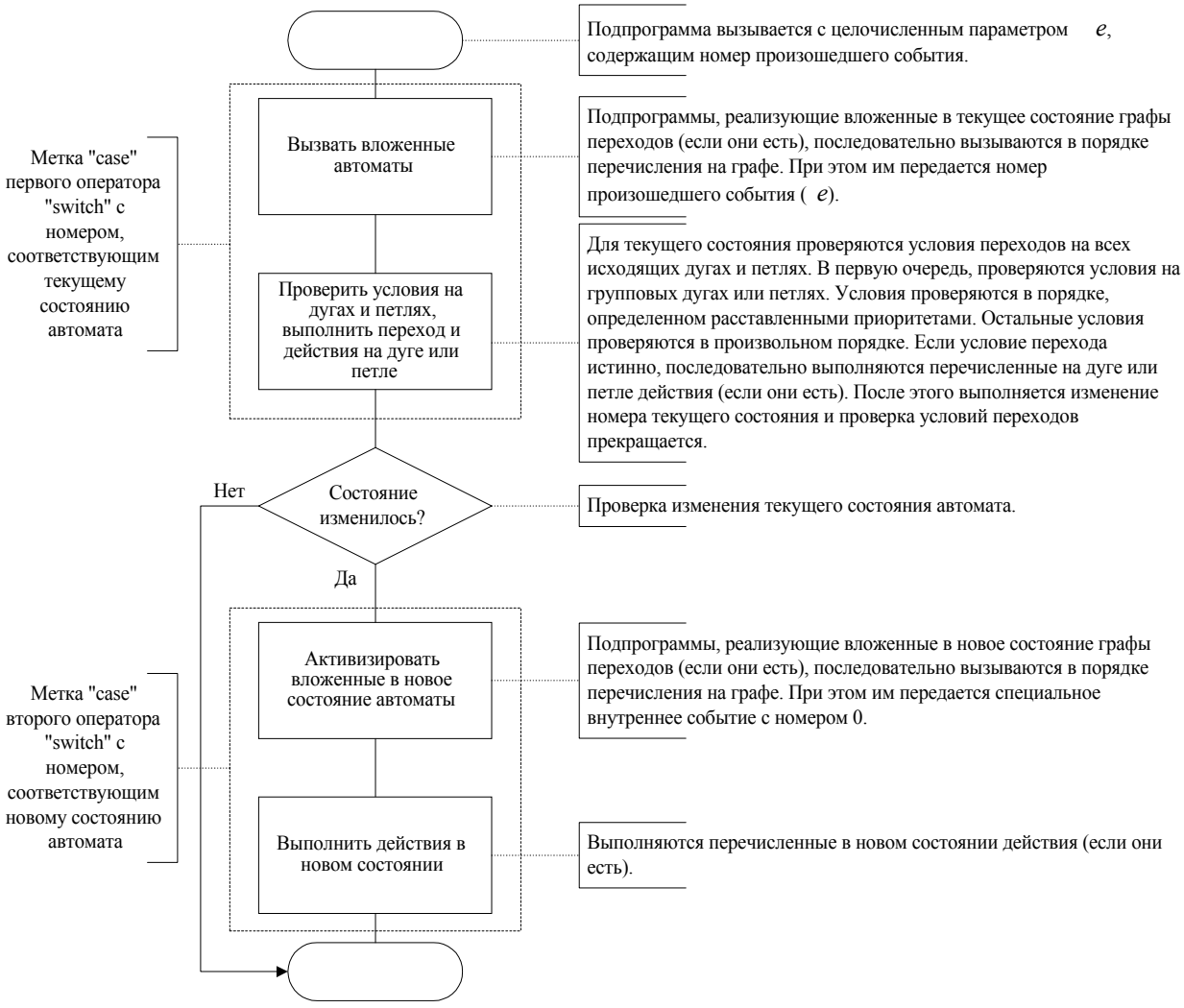
## 7.1. Нотация, используемая в графах переходов




Используемые обозначения

| | |
|---|---|
| An | автомат с номером n. |
| yn | переменная состояния автомата с номером n. |
| xj | входная переменная с номером j. |
| zk | выходное воздействие с номером k. |
| zd_m | m-ое значение выходного воздействия с номером d. |
| en | событие с номером n. В условиях переходов "en" является сокращенной записью предиката "e == n", а "¬en" - сокращенной записью предиката "e != n". |
| yn = p | условие перехода по номеру состояния автомата с номером n (предикат "yn == p"). |
| Cn_r | условие перехода с номером r для автомата с номером n. |
| i: | обозначение приоритета условия перехода (1 - наивысший приоритет). |

## 7.2. Схема универсального алгоритма реализации автоматов



## 7.3. Шаблон Си-функции, реализующей автомат

```
// Шаблон функции, реализующей автомат. Заменить "_i_" на номер автомата.
void A_i_( int e )
{
  int y_old = y_i_ ;

  // Протоколирование запуска автомата.
  #ifdef A_i__BEGIN_LOGGING
    log_begin( "A_i_", y_old, e ) ;
  #endif

  switch( y_i_ )
  {
    case 0:
      // Вызвать вложенные автоматы.
      // Проверить условия на дугах и петлях,
      // выполнить переход и действия на дуге или петле.
    break ;

    ...

    case n:
      // Вызвать вложенные автоматы.
      // Проверить условия на дугах и петлях,
      // выполнить переход и действия на дуге или петле.
    break ;

    default:
      #ifdef A_i__ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR,
                   "Ошибка в автомате A_i_: неизвестный номер состояния!", 0 ) ;
      #endif
  } ;

  // Если состояние не изменилось - завершить выполнение функции.
  if( y_old == y_i_ ) goto A_i__end ;

  // Протоколирование перехода в автомате.
  #ifdef A_i__TRANS_LOGGING
    log_trans( "A_i_", y_old ) ;
  #endif

  switch( y_i_ )
  {
    case 0:
      // Произвести активизацию вложенных в новое состояние автоматов.
      // Выполнить действия в новом состоянии.
    break ;

    ...

    case n:
      // Произвести активизацию вложенных в новое состояние автоматов.
      // Выполнить действия в новом состоянии.
    break ;
  } ;

  // Протоколирование завершения работы автомата.
  A_i__end: ;
  #ifdef A_i__END_LOGGING
    log_end( "A_i_", y_i_, e ) ;
  #endif
} ;
```

# 8. СИСТЕМОНЕЗАВИСИМАЯ ЧАСТЬ

## 8.1. Автомат контроля режима печати (A0)

### 8.1.1. Словесное описание

Автомат контроля режима печати предназначен для обеспечения предварительного просмотра и печати файлов в различных режимах работы системы. Автомат определяет поведение модуля управления печатью, являющегося составной частью системы и подключаемого к ней при компиляции.

Модуль печати реализует следующие функции:

- печать строк с информацией о происходящих в системе событиях в режиме автоматического функционирования системы;
- предварительный просмотр отдельных протоколов работы системы и их печать по требованию оператора;
- печать выбранных протоколов работы системы по требованию оператора или при срабатывании таймера.

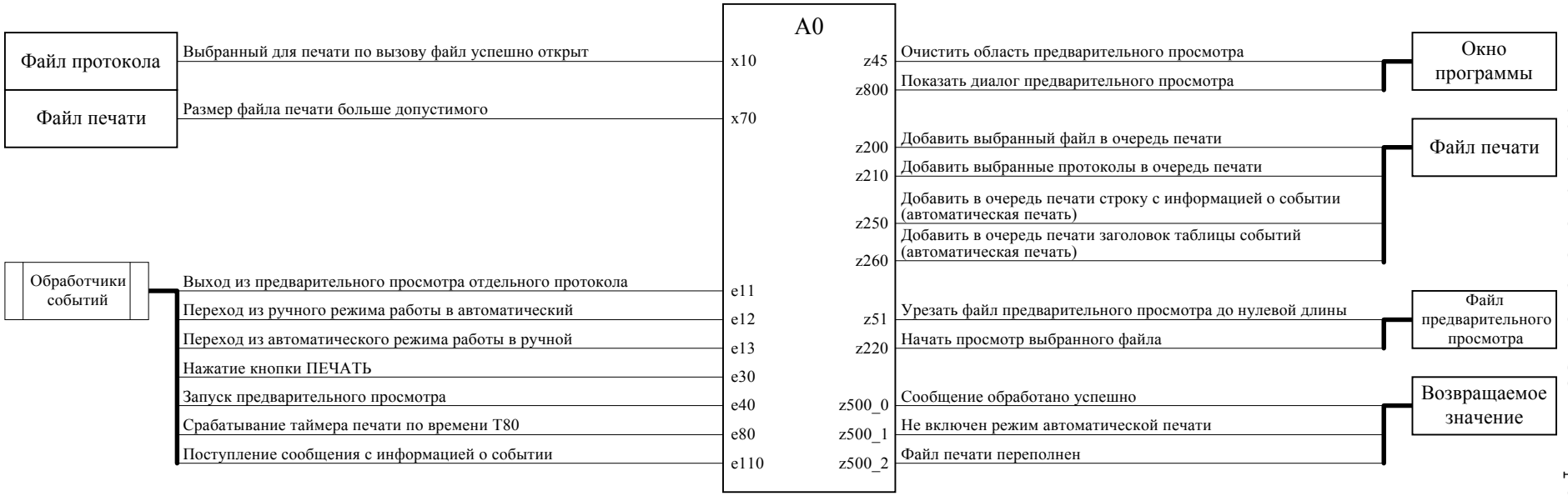
При переводе системы в автоматический режим работы (событие e12) автомат переходит в состояние "Автоматический", при этом в очередь печати добавляется заголовок таблицы происходящих в системе событий. Добавление в очередь печати строки с информацией о произошедшем в системе событии осуществляется при вызове интерфейсной функции print_on_event(), в качестве параметра которой передается указанная строка. Эта функция запускает автомат с событием, означающим необходимость напечатать очередную строку (событие e110). Если размер файла печати не превышает заданного (переменная x70), переданная строка добавляется в этот файл (являющийся очередью печати и обрабатываемый менеджером печати), а функция print_on_event() возвращает 0. В противном случае функция print_on_event() возвращает 2.

При переводе системы в ручной режим работы (событие e13), автомат переходит в состояние "Ручной общий", в котором при нажатии общей кнопки "ПЕЧАТЬ" (событие e30) или срабатывании таймера печати по времени (событие e80) протоколы работы системы в соответствии с текущим выбором, выполненным оператором на видеокадре протоколов, добавляются в очередь печати.

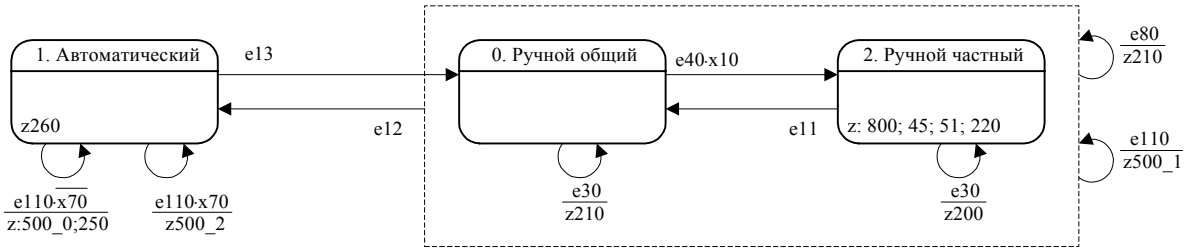
При нажатии оператором на видеокадре протоколов работы системы одной из кнопок, соответствующих различным протоколам, автомат вызывается с событием e40. При этом в качестве параметра функция автомата получает указатель на структуру данных с информацией о диалоге, в котором должен осуществляться предварительный просмотр выбранного протокола. После этого, при условии успешного открытия файла данного протокола (переменная x10), автомат переходит в состояние "Ручной частный", в котором открывается указанный диалог и инициируется предварительный просмотр выбранного протокола. При нажатии оператором в диалоге предварительного просмотра кнопки "ПЕЧАТЬ" (событие e30), выбранный протокол добавляется в очередь печати.

Возврат из состояния "Ручной частный" в состояние "Ручной общий" осуществляется при закрытии диалога предварительного просмотра (событие e11).

Автомат контроля режима печати. Схема связей автомата A0




Автомат контроля режима печати. Граф переходов A0



### 8.1.3. Текст функции, реализующей автомат

```
#include <sys/types.h>
#include "photon_stuff.h"
#include "log.h"
#include "defines.h"

static int y0 = 0 ;

void A0( int e, preview_dialog_t *d )
{
  int y_old = y0 ;

  #ifdef A0_BEGIN_LOGGING
    log_begin( "A0", y_old, e ) ;
  #endif

  switch( y0 )
  {
    case 0:
      if( e == 80 )  { z210() ; }
      else
      if( e == 110 ) { z500_1() ; }
      else
      if( e == 12 )                  y0 = 1 ;
      else
      if( e == 30 )  { z210() ; }
      else
      if( e == 40 && x10(d))         y0 = 2 ;
    break ;

    case 1:
    {
      int    X70 = x70() ;

      if( e == 13 )                  y0 = 0 ;
      else
      if( e == 110 && X70 ) { z500_2() ; }
      else
      if( e == 110 && !X70 ) { z500_0() ; z250() ; }
    }
    break ;

    case 2:
    {
      if( e == 80 )  { z210() ; }
      else
      if( e == 110 ) { z500_1() ; }
      else
      if( e == 12 )                  y0 = 1 ;
      else
      if( e == 11 )                  y0 = 0 ;
      else
      if( e == 30 )  { z200(d) ; }
    }
    break ;

    default:
      #ifdef A0_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, "ERROR IN A0: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == y0 ) goto A0_end ;

  {
    #ifdef A0_TRANS_LOGGING
      log_trans( "A0", y_old, y0 ) ;
    #endif
  } ;
```

```
  switch( y0 )
  {
    case 1:
      z260() ;
    break ;

    case 2:
      z800(d) ; z45(d) ; z51() ; z220(d) ;
    break ;
  } ;

  A0_end: ;
  #ifdef A0_END_LOGGING
    log_end( "A0", y0, e ) ;
  #endif
} ;
```

## 8.2. Автомат выполнения печати (A1)

### 8.2.1. Словесное описание

Автомат выполнения печати описывает поведение менеджера печати, функционирующего как отдельный процесс и предназначенного для непосредственной передачи на принтер распечатываемых документов.

Запуск автомата происходит при срабатывании таймера (событие e70), период которого задается в программе. Этот таймер запускается в режиме генератора синхроимпульсов при инициализации программы.

Автомат следит за двумя файлами:

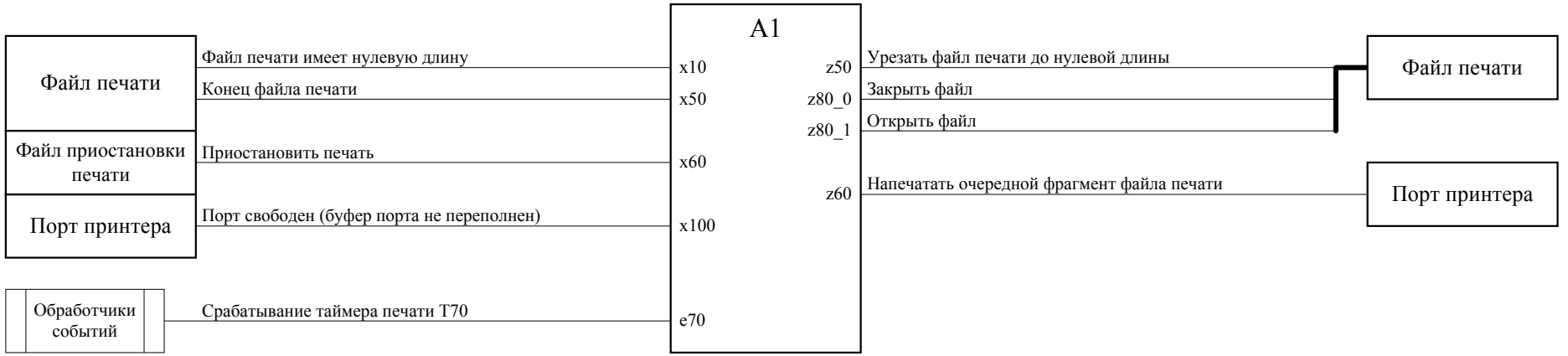
- файлом, наличие которого является сигналом приостановки печати (переменная x60);
- файлом печати, в который добавляются выдаваемые на печать документы (переменные x10, x50).

Так как эти файлы физически находятся в оперативной памяти (на ram-диске), периодическая проверка их состояния не должна заметно влиять на общую производительность системы, особенно учитывая то, что менеджер печати запускается с низким приоритетом, задаваемым в программе при инициализации.

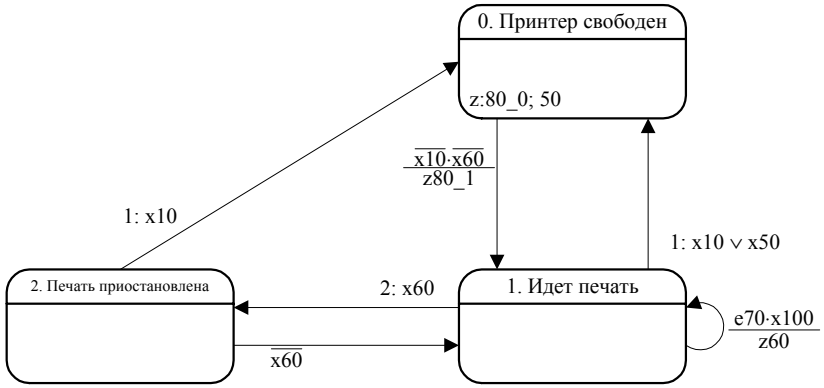
При отсутствии команды на приостановку печати (отсутствует файл приостановки печати) поведение автомата может быть описано следующим образом: при добавлении нового документа в файл печати, автомат переходит в состояние "Идет печать", в котором при срабатывании таймера в порт принтера отправляется очередной фрагмент распечатываемого документа. Размер этого фрагмента задается в программе и выбирается в зависимости от заданного периода срабатывания таймера и быстродействия принтера. При достижении конца файла печати автомат переходит в состояние "Принтер свободен", при этом файл печати урезается до нулевой длины. Если во время печати файл печати был урезан до нулевой длины, это является командой прекращения печати, после чего автомат также переходит в состояние "Принтер свободен".

Для устранения возможности неправильной работы, связанной с одновременной модификацией файла печати менеджером печати и другими процессами, используется механизм блокировки файлов.

Автомат выполнения печати. Схема связей автомата A1




Автомат выполнения печати. Граф переходов A1



### 8.2.3. Текст функции, реализующей автомат

```
#include <sys/types.h>
#include "log.h"
#include "defines.h"
#include "x.h"
#include "z.h"

static int y1 = 0 ;

void A1( int e )
{
  int y_old = y1 ;

  #ifdef A1_BEGIN_LOGGING
    log_begin( "A1", y_old, e ) ;
  #endif

  switch( y1 )
  {
    case 0:
      if( !x10() && !x60() )  { z80_1() ;            y1 = 1 ; } ;
    break ;

    case 1:
      if( x10() || x50() )                          y1 = 0 ;
      else
      if( x60() )                                   y1 = 2 ;
      else
      if( e == 70 && x100() ) { z60() ; }
    break ;

    case 2:
      if( x10() )                                   y1 = 0 ;
      else
      if( !x60() )                                  y1 = 1 ;
    break ;

    default:
      #ifdef A1_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, "ERROR IN A1: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == y1 ) goto A1_end ;

  {
    #ifdef A1_TRANS_LOGGING
      log_trans( "A1", y_old, y1 ) ;
    #endif
  } ;

  switch( y1 )
  {
    case 0:
      z80_0() ; z50() ;
    break ;
  } ;

  A1_end: ;
  #ifdef A1_END_LOGGING
    log_end( "A1", y1, e ) ;
  #endif
} ;
```

## 8.3. Автомат буферизации предварительного просмотра (A2)

### 8.3.1. Словесное описание

Автомат буферизации предварительного просмотра предназначен для ускорения графического вывода при выполнении предварительного просмотра протоколов работы системы. Автомат вызывается в функции перерисовки окна предварительного просмотра.

Необходимость в разработке нестандартных средств просмотра файла объясняется тем, что в используемом для предварительного просмотра графическом окне невозможно при помощи функций рисования текста отобразить символы псевдографики, которые могут встречаться в просматриваемых файлах. Поэтому используемые символы псевдографики отображаются при помощи функций рисования линий.

Для ускорения перерисовки окна предварительного просмотра содержимое просматриваемого файла выводится, по возможности, блоками. При этом:

- непрерывные последовательности текстовых символов, не содержащие пробелов, выводятся одним блоком. Корректно выводить последовательности текстовых символов, содержащие пробелы, невозможно из-за ошибки в функции отрисовки текста, проявляющейся в том, что при отрисовке ширина символа "пробел" не совпадает с шириной обычного текстового символа;
- непрерывные последовательности пробелов не отрисовываются;
- непрерывные последовательности псевдографических символов "горизонтальная линия" отрисовываются одним вызовом функции рисования линии;
- другие символы псевдографики отрисовываются посимвольно.

Автомат выполняет функцию распознавателя указанных последовательностей.

Состояние "Вывод символа" соответствует посимвольному выводу.

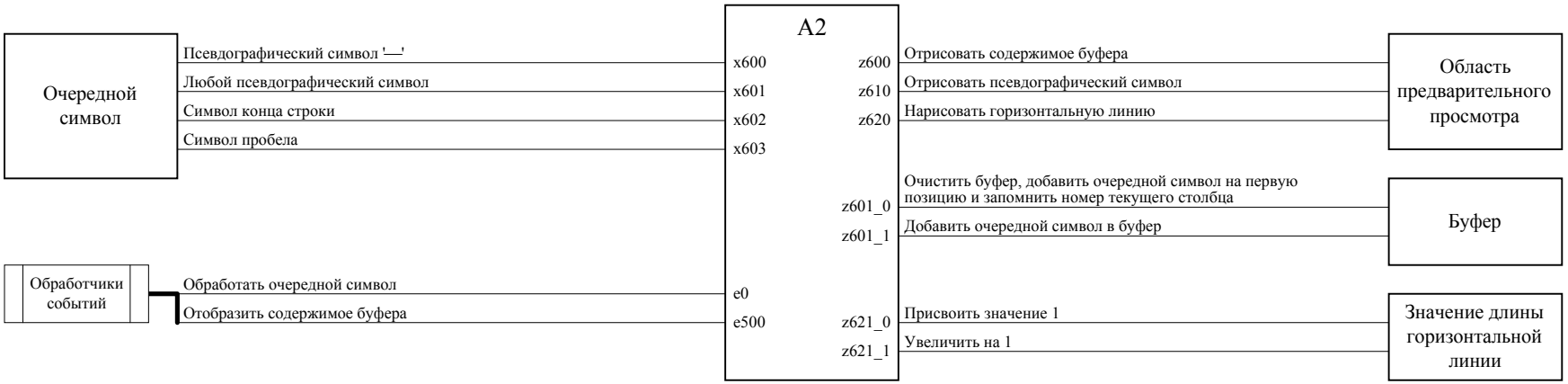
Состояние "Текст" соответствует накоплению непрерывной последовательности текстовых символов.

Состояние "Линия" соответствует накоплению непрерывной последовательности псевдографических символов "горизонтальная линия".

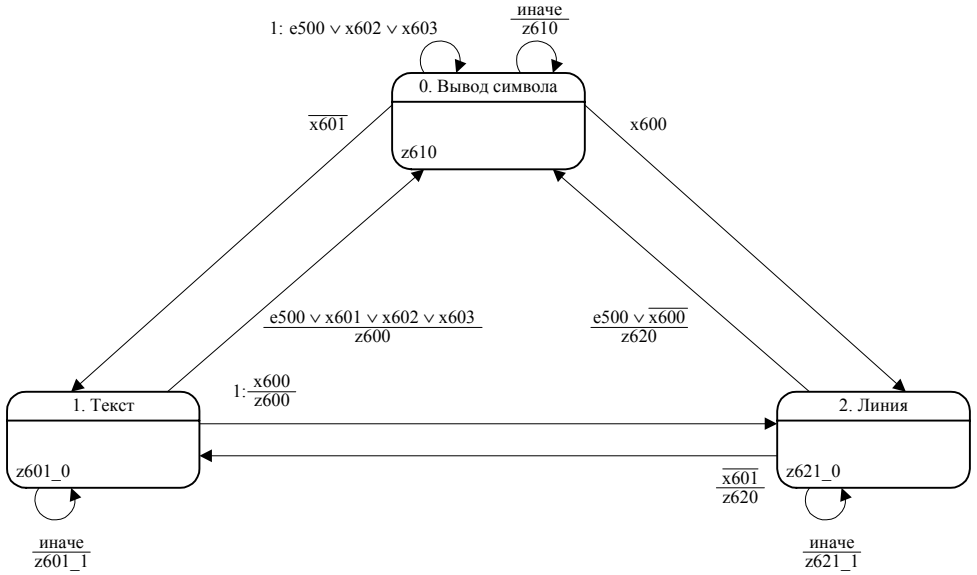
Для обработки очередного символа автомат вызывается с событием `e0`, а при завершении обработки видимой части файла автомат вызывается с событием `e500`, по которому происходит отрисовка накопленного в буфере содержимого. Кроме номера события автомату также передается код обрабатываемого символа и номера строки и столбца, соответствующие его местоположению.

Для ускорения работы некоторые функции входных переменных и выходных воздействий реализованы внутри функции автомата.

Автомат буферизации предварительного просмотра. Схема связей автомата A2




Автомат буферизации предварительного просмотра. Граф переходов A2



### 8.3.3. Текст функции, реализующей автомат

```
#include "photon_stuff.h"
#include "log.h"
#include "defines.h"

void A2( int e, char c, int line_num, int col_num )
{
  static int    y2 = 0 ;
  int           y_old = y2 ;

  static char   buf[line_width+10] = "" ;
  static int    buf_len = 0 ;
  static int    line_len = 0 ;
  static int    start_col = 0 ;

  #ifdef A2_SHOW_SYMBOL
  {
    char    str[100] = "" ;

    sprintf( str, "Получен символ %d (`%c`)", c, c ) ;
    log_write( '!', str, 0 ) ;
  }
  #endif

  #ifdef A2_BEGIN_LOGGING
    log_begin( "A2", y_old, e ) ;
  #endif

  switch( y2 )
  {
    case 0:
      if( e == 500 || (c == 0x0A || c == 0x0D) || (c == ' ')) ;
      else
      if( c < 176 || c > 223 )                                  y2 = 1 ;
      else
      if( c == '-' )                                            y2 = 2 ;
      else
      { z610( c, line_num, col_num ) ; }
    break ;

    case 1:
      if( c == '-' )
      { z600( line_num, col_num, buf, buf_len, start_col ) ;    y2 = 2 ; }
      else
      if( e == 500 || (c >= 176 && c <= 223) || (c == 0x0A || c == 0x0D) || (c == ' ') )
        { z600( line_num, col_num, buf, buf_len, start_col ) ;  y2 = 0 ; }
      else
      { buf[buf_len++] = c ; buf[buf_len] = 0 ; }

    break ;

    case 2:
      if( e == 500 || c != '-' )
      { z620( line_num, col_num, line_len, start_col ) ;        y2 = 0 ; }
      else
      if( c < 176 || c > 223 )
      { z620( line_num, col_num, line_len, start_col ) ;        y2 = 1 ; }
      else
      { line_len++ ; }
    break ;

    default:
      #ifdef A2_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, "ERROR IN A2: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == y2 ) goto A2_end ;

    #ifdef A2_TRANS_LOGGING
      log_trans( "A2", y_old, y2 ) ;
    #endif
```

```
  switch( y2 )
  {
    case 0:
      z610( c, line_num, col_num ) ;
    break ;

    case 1:
      start_col = col_num ;
      buf[0] = c ; buf_len = 1 ; buf[1] = 0 ;
    break ;

    case 2:
      start_col = col_num ;
      line_len = 1 ;
    break ;
  } ;

  A2_end: ;
  #ifdef A2_END_LOGGING
    log_end( "A2", y2, e ) ;
  #endif
}
```

# 9. ИСХОДНЫЕ ТЕКСТЫ СИСТЕМОЗАВИСИМОЙ ЧАСТИ

## 9.1. Модуль печати

### 9.1.1. Входные переменные (файл x.c)

```
//=====================================================================
//
// Модуль содержит функции, реализующие входные переменные.
//
//

#include "photon_stuff.h"
#include "defines.h"
#include "log.h"
#include <sys/dev.h>

extern char     print_preview_file_name[] ;
extern char     print_spool_file_name[]  ;
extern char     print_current_dir[] ;
extern int      print_preview_file_limit ;
extern int      print_spool_file_limit ;

int x10( preview_dialog_t *d )
{
  int       result ;
  FILE      *print_preview_file = NULL ;

  if( d->file_name )
  {
    print_preview_file = fopen( d->file_name, "r" ) ;
    result = print_preview_file != NULL ;
    fclose( print_preview_file ) ;
  }
  else
    result = 0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x10 - выбранный для печати файл существует", result ) ;
  #endif

  return result ;
} ;

int x70()
{
  int           result = 1 ;

  result = print_file_size( print_spool_file_name ) > print_spool_file_limit ;

  #ifdef INPUTS_LOGGING
    log_input( "x70 - размер файла печати больше допустимого", result ) ;
  #endif

  return result ;
} ;

int x600( char c )
{
  int           result = 1 ;

  result = c == '-' ;

  #ifdef INPUTS_LOGGING
    log_input( "x600 - получен псевдографический символ горизонтальной линии", result ) ;
  #endif

  return result ;
} ;
```

```
int x601( char c )
{
  int           result = 1 ;

  result = c >= 176 && c <= 223 ;

  #ifdef INPUTS_LOGGING
    log_input( "x601 - получен псевдографический символ", result ) ;
  #endif

  return result ;
} ;

int x602( char c )
{
  int           result = 1 ;

  result = c == 0x0A || c == 0x0D ;

  #ifdef INPUTS_LOGGING
    log_input( "x602 - получен символ конца строки", result ) ;
  #endif

  return result ;
} ;

int x603( char c )
{
  int           result = 1 ;

  result = c == ' ' ;

  #ifdef INPUTS_LOGGING
    log_input( "x603 - получен символ конца пробел", result ) ;
  #endif

  return result ;
} ;
```

### 9.1.2. Выходные воздействия (файл z.c)

```
//========================================================================
//
// Модуль содержит функции, реализующие выходные воздействия.
//
//

#include "photon_stuff.h"
#include "log.h"
#include "defines.h"
#include "nls_api.h"
#include <sys/kernel.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <share.h>

extern char         print_suspend_semaphor_file_name[] ;
extern char         print_spool_file_name[] ;
extern char         print_preview_file_name[] ;
extern int          print_buffer_size ;
extern int          print_on_event_result ;
extern PhRect_t     symbol_size ;
extern PhPoint_t    symbol_center ;

extern preview_dialog_t *active_dialog ;
extern preview_dialog_t dialogs[] ;

void z45( preview_dialog_t *d )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z45. Очистить окно предварительного просмотра.", 0 ) ;
  #endif

  print_preview_stop_preview( d ) ;
} ;
```

```
void z50()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z50. Урезать файл спулинга.", 0 ) ;
  #endif

  {
    int     filedes = -1 ;

    filedes = sopen( print_spool_file_name, O_WRONLY | O_CREAT | O_TRUNC,
                     SH_DENYWR, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP ) ;

    if( filedes != -1 )
      close( filedes ) ;
   }
} ;

void z51()
{
  FILE    *file ;

  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z51. Урезать файл предварительного просмотра.", 0 ) ;
  #endif

  file = fopen( print_preview_file_name, "w" ) ;
  if( file )
    fclose( file ) ;
} ;

void z55_0()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z55_0. Disable suspending.", 0 ) ;
  #endif

  remove( print_suspend_semaphor_file_name ) ;
} ;

void z55_1()
{
  struct stat   buf ;

  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z55_1. Enable or disable suspending.", 0 ) ;
  #endif

  if( stat( print_suspend_semaphor_file_name, &buf ) == 0 )
  {
    // Suspend file exists.
    remove( print_suspend_semaphor_file_name ) ;
  }
  else
  {
    FILE    *file ;

    file = fopen( print_suspend_semaphor_file_name, "w" ) ;
    if( file )
      fclose( file ) ;
  }
} ;

void z200( preview_dialog_t *d )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z200. Добавить выбранный файл в очередь печати.", 0 ) ;
  #endif

  print_files_protocol_print( d->file_name ) ;
} ;
```

```
void z210()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z210. Добавить выбранные протоколы в очередь печати.", 0 ) ;
  #endif

  print_files_protocols_print() ;
} ;

void z220( preview_dialog_t *d )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z220. Начать просмотр выбранного файла.", 0 ) ;
  #endif

  print_files_protocol_preview( d->file_name ) ;
  print_preview_begin_preview( d ) ;
} ;

void z250()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z250. Напечатать строку с информацией о событии.", 0 ) ;
  #endif

  print_files_event_print() ;
} ;

void z260()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z260. Напечатать заголовок таблицы событий.", 0 ) ;
  #endif

  print_files_event_header_print() ;
} ;

void z500_0()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z500_0. Ответ клиенту - сообщение обработано успешно.", 0 ) ;
  #endif

  print_on_event_result = 0 ;
} ;

void z500_1()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION,
               "z500_1. Ответ клиенту - сервер не в режиме автоматической печати.", 0 ) ;
  #endif

  print_on_event_result = 1 ;
} ;

void z500_2()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z500_2. Ответ клиенту - файл печати переполнен.", 0 ) ;
  #endif

  print_on_event_result = 2 ;
} ;
```

```
void z600( int line_num, int col_num, const char * const buf, int buf_len,
           int start_col )
{
  #ifdef A2_ACTIONS_LOGGING
    log_write( LOG_ACTION, "z600. Вывести буфер предварительного просмотра.", 0 ) ;
  #endif

  {
    PhPoint_t       pos ;
    char            str2[(line_width+10)*2] = "" ;

    pos.x = start_col * symbol_size.lr.x ;
    pos.y = (line_num+1) * symbol_size.lr.y - 4 ;

    UTF_2( str2, buf ) ;

    PgDrawString( str2, &pos ) ;
  }
} ;

void z601_0( char c, const char * const buf, int *buf_len )
{
  #ifdef A2_ACTIONS_LOGGING
    log_write( LOG_ACTION, "z601_0. Добавить символ в первую позицию буфера.", 0 ) ;
  #endif

  buf[0] = c ; *buf_len = 1 ; buf[1] = 0 ;
} ;

void z601_1( char c, const char * const buf, int *buf_len )
{
  #ifdef A2_ACTIONS_LOGGING
    log_write( LOG_ACTION, "z601_0. Добавить символ в буфер.", 0 ) ;
  #endif

  buf[*buf_len] = c ; *buf_len = *buf_len + 1 ; buf[*buf_len] = 0 ;
} ;

void z610( char c, int line_num, int col_num )
{
  PhPoint_t       pos ;

  #ifdef A2_ACTIONS_LOGGING
    log_write( LOG_ACTION, "z610. Вывести символ псевдографики.", 0 ) ;
  #endif

  pos.x = col_num * symbol_size.lr.x ;
  pos.y = (line_num+1) * symbol_size.lr.y ;

  switch( c )
  {
    case '│':
      PgDrawILine(
                   pos.x + symbol_center.x,    pos.y - symbol_size.lr.y,
                   pos.x + symbol_center.x,    pos.y
                 ) ;
    break ;

    case '├':
      PgDrawILine(
                   pos.x + symbol_center.x,    pos.y - symbol_center.y,
                   pos.x + symbol_size.lr.x,   pos.y - symbol_center.y
                 ) ;
      PgDrawILine(
                   pos.x + symbol_center.x,    pos.y - symbol_size.lr.y,
                   pos.x + symbol_center.x,    pos.y
                 ) ;
    break ;



    case '┬':
      PgDrawILine(
                   pos.x,                      pos.y - symbol_center.y,
                   pos.x + symbol_size.lr.x,   pos.y - symbol_center.y
                 ) ;
```

```
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_center.y,
              pos.x + symbol_center.x,    pos.y
            ) ;
break ;

case '┤':
  PgDrawILine(
              pos.x,                      pos.y - symbol_center.y,
              pos.x + symbol_center.x,    pos.y - symbol_center.y
            ) ;
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_size.lr.y,
              pos.x + symbol_center.x,    pos.y
            ) ;
break ;

case '┴':
  PgDrawILine(
              pos.x,                      pos.y - symbol_center.y,
              pos.x + symbol_size.lr.x,   pos.y - symbol_center.y
            ) ;
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_size.lr.y,
              pos.x + symbol_center.x,    pos.y - symbol_center.y
            ) ;
break ;

case '┼':
  PgDrawILine(
              pos.x,                      pos.y - symbol_center.y,
              pos.x + symbol_size.lr.x,   pos.y - symbol_center.y
            ) ;
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_size.lr.y,
              pos.x + symbol_center.x,    pos.y
            ) ;
break ;

case '┌':
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_center.y,
              pos.x + symbol_size.lr.x,   pos.y - symbol_center.y
            ) ;
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_center.y,
              pos.x + symbol_center.x,    pos.y
            ) ;
break ;

case '┐':
  PgDrawILine(
              pos.x,                      pos.y - symbol_center.y,
              pos.x + symbol_center.x,    pos.y - symbol_center.y
            ) ;
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_center.y,
              pos.x + symbol_center.x,    pos.y
            ) ;
break ;

case '┘':
  PgDrawILine(
              pos.x,                      pos.y - symbol_center.y,
              pos.x + symbol_center.x,    pos.y - symbol_center.y
            ) ;
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_size.lr.y,
              pos.x + symbol_center.x,    pos.y - symbol_center.y
            ) ;
break ;

case '└':
  PgDrawILine(
              pos.x + symbol_center.x,    pos.y - symbol_center.y,
              pos.x + symbol_size.lr.x,   pos.y - symbol_center.y
            ) ;
```

```
      PgDrawILine(
                  pos.x + symbol_center.x,    pos.y - symbol_size.lr.y,
                  pos.x + symbol_center.x,    pos.y - symbol_center.y
                ) ;
    break ;
  }
} ;

void z620( int line_num, int col_num, int line_len, int start_col )
{
  PhPoint_t       pos ;

  #ifdef A2_ACTIONS_LOGGING
    log_write( LOG_ACTION, "z620. Нарисовать горизонтальную линию.", 0 ) ;
  #endif

  pos.x = start_col * symbol_size.lr.x ;
  pos.y = (line_num+1) * symbol_size.lr.y ;

  PgDrawILine(
              pos.x,
              pos.y - symbol_center.y,
              pos.x + symbol_size.lr.x * line_len,
              pos.y - symbol_center.y
            ) ;
} ;

void z621_0( int *line_len )
{
  #ifdef A2_ACTIONS_LOGGING
    log_write( LOG_ACTION, "z621_0. Присвоить значению длины линии 1.", 0 ) ;
  #endif

  *line_len = 1 ;
} ;

void z621_1( int *line_len )
{
  #ifdef A2_ACTIONS_LOGGING
    log_write( LOG_ACTION, "z621_1. Увеличить значение длины линии.", 0 ) ;
  #endif

  *line_len = *line_len + 1 ;
} ;

void z800( preview_dialog_t *d )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z800. Показать окно предварительного просмотра.", 0 ) ;
  #endif

  d->abw = d->ap_base[d->dialog].wgt ;
  d->preview_wgt = d->ap_base[d->preview].wgt ;
  d->scrollbar_wgt = d->ap_base[d->scrollbar].wgt ;

  PtWindowToFront( d->abw ) ;
} ;
```

### 9.1.3. Обработчики событий нажатия кнопок (файл print_buttons.c, частично)

```
#include "photon_stuff.h"

extern preview_dialog_t     dialogs[] ;
extern preview_dialog_t     *active_dialog ;

int print_buttons( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  if(PtWidgetParent(ABW_prot_anpar)!=NULL)
  {
   active_dialog = &dialogs[0] ;
    A0( 40, active_dialog ) ;
  }
  else
```

```
  if(PtWidgetParent(ABW_prot_ost)!=NULL)
  {
    active_dialog = &dialogs[1] ;
    A0( 40, active_dialog ) ;
  }
  else
  //...
  if(PtWidgetParent(ABW_kzip)!=NULL)
  {
    active_dialog = &dialogs[31];
    A0( 40, active_dialog ) ;
  }
  else
  if( widget == ABW_print_clear_spool )
  {
    z50() ;
  }
  else
  if( widget == ABW_print_suspend_printing)
  {
    z55_1() ;
  }

  if( widget == ABW_F6)
  {
    A0( 30, active_dialog ) ;
  }

  return( Pt_CONTINUE );
}
```

### 9.1.4. Обработчик события закрытия диалога (файл preview_close.c)

```
#include "photon_stuff.h"

extern preview_dialog_t     *active_dialog ;

int preview_close( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  active_dialog = NULL ;
  A0( 11, NULL ) ;

  return( Pt_CONTINUE ) ;
} ;
```

### 9.1.5. Обработчик события срабатывания таймера (файл print_timers.c)

```
#include "photon_stuff.h"

int T80( void *data, pid_t pid, void *msg, size_t msg_size )
{
  A0( 80, NULL ) ;

  return Pt_CONTINUE ;
} ;
```

### 9.1.6. Модуль инициализации программы (файл setup1.c, частично)

```
//=====================================================================
//
// Инициализация главного окна программы.
//

#include "photon_stuff.h"

//*********************************************************************
//*
//* Параметры для настройки менеджера печати.
//*

// Имя временного файла для предварительного просмотра (с полным путем).
const char          print_preview_file_name[PATH_MAX] = "/hd/print.preview" ;
```

```
// Имя временного файла для печати - файла спулинга (с полным путем).
const char          print_spool_file_name[PATH_MAX] = "/hd/print.spool" ;
const char          print_suspend_semaphor_file_name[PATH_MAX] = "/hd/print.suspend" ;

// Имя файла, содержащего заголовок таблицы для печати в автоматическом режиме
// (с полным путем).
const char          print_preview_event_header_file_name[PATH_MAX] = "/hd/pr_tc.txt" ;

// Путь к каталогу, из которого будут выбираться файлы в режиме печати по вызову.
const char          print_current_dir[PATH_MAX] = "/hd" ;

// Имя шрифта, используемого для предварительного просмотра.
const char          preview_font[] = "courc12b" ;

// Цвет символов, используемый для предварительного просмотра.
const PgColor_t     preview_color = Pg_BLACK ;

// Максимально допустимый размер временного файла для предварительного просмотра.
const int           print_preview_file_limit = 50000 ;

// Максимально допустимый размер временного файла для печати.
const int           print_spool_file_limit = 200000 ;

//*
//*
//*********************************************************************

// Строка с информацией о событии.
char                print_event_string[line_width+10] = "" ;

// Результат обработки приема строки в режиме автоматической печати.
int                 print_on_event_result = 0 ;

preview_dialog_t    dialogs[40] ;
preview_dialog_t    *active_dialog = NULL ;

//=============================================================================================
// Функция инициализации базового окна программы.
//

int setup1( PtWidget_t *link_instance, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  // Произвести инициализацию.
  memset( dialogs, 0, sizeof(dialogs)) ;

  //* Значение поля apbase берется из файла abwidgets.h.
  dialogs[0].abm = ABM_prot_anpar ;
  dialogs[0].ap_base = dsprot_anpar ;
  dialogs[0].dialog = ABN_prot_anpar ;
  dialogs[0].preview = ABN_preview_anpar ;
  dialogs[0].scrollbar = ABN_scroll_anpar ;
  dialogs[0].selected = 1 ;
  strcpy( dialogs[0].file_name, "//0/hd/pr_anpar.txt" ) ;

  dialogs[1].abm = ABM_prot_ost;
  dialogs[1].ap_base = dsprot_ost ;
  dialogs[1].dialog = ABN_prot_ost ;
  dialogs[1].preview = ABN_preview_ost ;
  dialogs[1].scrollbar = ABN_scroll_ost ;
  dialogs[1].selected = 1 ;
  strcpy( dialogs[1].file_name, "//0/hd/pr_ost.txt" ) ;

  //...

  dialogs[31].abm = ABM_kzip ;
  dialogs[31].ap_base = dskzip ;
  dialogs[31].dialog = ABN_kzip ;
  dialogs[31].preview = ABN_preview_zip ;
  dialogs[31].scrollbar = ABN_scroll_zip ;
  dialogs[31].selected = 0 ;
  strcpy( dialogs[31].file_name, "//0/hd/zip1.txt") ;

  z51() ;
  z55_0() ;

  return( Pt_CONTINUE );
}
```

```
//======
// Интерфейсная функция вывода строки в режиме автоматической печати.
// Возвращает:
//  0 - строка отправлена на печать;
//  1 - не включен автоматический режим печати;
//  2 - файл печати переполнен.
//
int print_on_event( const char * const str )
{
  print_event_string[0] = 0 ;
  strncpy( print_event_string, str, line_width ) ;
  print_event_string[line_width] = 0 ;

  A0(110, NULL ) ;

  return print_on_event_result ;
}
```

### 9.1.7. Функции манипулирования файлами (файл print_files.c)

```
//=====================================================================
//
// Функции манипулирования файлами для менеджера печати.
//
//
#include "photon_stuff.h"
#include <sys/stat.h>
#include <sys/timeb.h>
#include <time.h>

extern char     print_preview_file_name[] ;
extern char     print_spool_file_name[] ;
extern char     print_event_string[] ;
extern char     print_preview_event_header_file_name[] ;

extern preview_dialog_t dialogs[] ;

int             print_preview_event_lines = 0 ;
int             print_spool_event_lines = 0 ;

//======
// Вернуть размер файла в байтах.
//
int print_file_size( const char * const name )
{
  struct stat   buf ;

  stat( name, &buf ) ;

  return buf.st_size ;
}

//======
// Добавить строку с текущей датой и временем в конец файла.
//
int print_file_time_append( const char * const dst_name )
{
  time_t        time_of_day ;
  char          str[100] = "" ;

  time_of_day = time( NULL ) ;
  strftime( str, 99, "%n%n%d.%m.%Y, %H:%M:%S%n", localtime( &time_of_day )) ;

  return 1 ;
}
```

```
//======
// Добавить строку в конец файла.
//
int print_file_strcopy_append( const char * const dst_name, const char * const str )
{
  FILE  *dst ;

  if( ! (dst = fopen( dst_name, "a" )) ) return 0 ;

  fwrite( str, strlen(str), 1, dst )  ;
  fclose( dst ) ;

  return 1 ;
}

//======
// Скопировать один файл в конец другого.
// Вставлять заголовок в начале каждого листа.
// Вернуть количество записанных строк,
// либо -1 в случае ошибки.
//
int print_file_copy_append( const char * const dst_name, const char * const src_name,
                            int draw_pagebreaks )
{
  char          c = 0x0A ;
  FILE          *src, *dst ;
  int           line_number = 0 ;
  int           page_number = 0 ;
  time_t        time_of_day ;
  char          str[100] = "" ;
  char          str1[1000] = "" ;

  if( ! (src = fopen( src_name, "r" )) ) return -1 ;
  if( ! (dst = fopen( dst_name, "a" )) ) return -1 ;

  time_of_day = time( NULL ) ;
  strftime( str, 99, "%d.%m.%Y, %H:%M:%S", localtime( &time_of_day )) ;

  while( !feof(src) )
  {
    if( c == 0x0A )
    {
      line_number++ ;

      if( draw_pagebreaks )
      if( page_number == 0 || (line_number % lines_on_page ) == 0 )
      {
        page_number++ ;
        line_number++ ;
        sprintf( str1, "--- %s (Лист %u): %s ----------------------------------------
----------------------------------\n"
                 , strrchr( src_name, '/' ) + 1, page_number, str ) ;

        str1[line_width+1] = 0 ;
        str1[line_width] = '\n' ;
        fwrite( str1, strlen(str1), 1, dst ) ;
      }
    }

    fwrite( &c, fread( &c, 1, 1, src ), 1, dst ) ;
  }

  fclose( src ) ;
  fclose( dst ) ;

  return line_number ;
}

//======
// Скопировать выбранный файл во временный файл печати.
//
int print_files_protocol_print( const char * const file_name )
{
  print_file_copy_append( print_spool_file_name, file_name, 1 ) ;

  return 1 ;
}
```

```
//======
// Скопировать выбранные протоколы во временный файл печати.
//
int print_files_protocols_print()
{
  int   i = 0 ;

  while( dialogs[i].abm )
  {
    if( dialogs[i].selected )
      print_file_copy_append( print_spool_file_name, dialogs[i].file_name, 1 ) ;

    i++ ;
  }

  return 1 ;
}

//======
// Скопировать выбранный файл во временный файл предварительного просмотра.
//
int print_files_protocol_preview( const char * const file_name )
{
  print_file_copy_append( print_preview_file_name, file_name, 1 ) ;
  return 1 ;
}

//======
// Добавить в конец файла разделитель страниц.
//
int print_files_event_pagebreak( const char * const dst_name, int line_num )
{
  time_t        time_of_day ;
  char          str[100] = "" ;
  char          str1[1000] = "" ;

  time_of_day = time( NULL ) ;
  strftime( str, 99, "%d.%m.%Y, %H:%M:%S", localtime( &time_of_day )) ;
  sprintf( str1, "--- Автоматическая печать (Лист %u): %s ------------------------------
-----------------------------------------------\n"
           , line_num / lines_on_page + 1, str ) ;

  str1[line_width+1] = 0 ;
  str1[line_width] = '\n' ;

  return print_file_strcopy_append( dst_name, str1 ) ;
}

//======
// Скопировать заголовок таблицы событий во временный файл печати.
//
int print_files_event_header_print()
{
  print_files_event_pagebreak( print_spool_file_name, print_spool_event_lines ) ;
  print_spool_event_lines =
   print_file_copy_append( print_spool_file_name, print_preview_event_header_file_name, 0
) ;

  return 1 ;
}

//======
// Скопировать строку с информацией о событии во временный файл печати.
//
int print_files_event_print()
{
  print_file_strcopy_append( print_spool_file_name, print_event_string ) ;
  print_spool_event_lines++ ;
  if( (print_spool_event_lines % lines_on_page) == 0 )
  {
    print_files_event_pagebreak( print_spool_file_name, print_spool_event_lines ) ;
    print_spool_event_lines++ ;
  }

  return 1 ;
}
```

```
//======
// Return total number of pages, printed in automatic mode.
//
int print_on_event_pages()
{
  return print_event_lines / lines_on_page + 1 ;
}

//======
// Return total number of lines, printed in automatic mode.
//
int print_on_event_lines()
{
  return print_event_lines ;
}
```

### 9.1.8. Функции, реализующие предварительный просмотр (файл print_preview.c)

```
//=====================================================================
//
// Функции, реализующие предварительный просмотр.
//

#include "photon_stuff.h"
#include <photon/PhRender.h>

extern char          print_preview_file_name[] ;
extern char          preview_font[] ;
extern PgColor_t     preview_color ;

extern preview_dialog_t *active_dialog ;

PhRect_t             symbol_size ;          // Размер символов для выбранного шрифта.
PhPoint_t            symbol_center ;        // Центральная точка символов.
static int           lines_in_file = 0 ;    // Количество строк в файле
                                            //  предварительного просмотра.
static int           lines_on_screen = 0 ;  // Количество строк, умещающихся на экран.
static char          str[1000] = "" ;

static int           print_preview_is_shown = 0 ; // Запущен ли предварительный просмотр.
static FILE          *print_preview_file = NULL ; // Файл для предварительного просмотра.

//======
// Перечитать файл предварительного просмотра.
//
int print_preview_reload_preview( preview_dialog_t *d )
{
  PtArg_t       args[1] ;

  print_preview_is_shown = 0 ;
  fclose( print_preview_file ) ;
  print_preview_file = NULL ;

  if( !print_setup_preview( d ) ) return 0 ;

  PtSetArg( args, Pt_ARG_SCROLL_POSITION, lines_in_file, 0 ) ;
  PtSetResources( d->scrollbar_wgt, 1, args ) ;

  print_preview_is_shown = 1 ;
  PtContainerGiveFocus( d->scrollbar_wgt, NULL ) ;
  PtDamageWidget( d->preview_wgt ) ;

  return 1 ;
} ;
```

```
//======
// Запустить предварительный просмотр.
//
int print_preview_begin_preview( preview_dialog_t *d )
{
  if( !print_setup_preview( d ) ) return 0 ;

  print_preview_is_shown = 1 ;
  PtContainerGiveFocus( d->scrollbar_wgt, NULL ) ;
  PtDamageWidget( d->preview_wgt ) ;

  return 1 ;
} ;

//======
// Прекратить предварительный просмотр.
//
int print_preview_stop_preview( preview_dialog_t *d )
{
  print_preview_is_shown = 0 ;
  PtDamageWidget( d->preview_wgt ) ;

  fclose( print_preview_file ) ;
  print_preview_file = NULL ;

  return 1 ;
} ;

//======
// Обработчик события перемещения скролбара.
//
int print_preview_scroll( PtWidget_t *widget, ApInfo_t *apinfo,
                          PtCallbackInfo_t *cbinfo )
{
  PtDamageWidget( active_dialog->preview_wgt ) ;

  return( Pt_CONTINUE ) ;
} ;

//======
// Функция перерисовки предварительного просмотра.
//
void print_preview_redraw( PtWidget_t *widget, PhTile_t *damage )
{
  PhRect_t      canvas ;

  PtSuperClassDraw( PtBasic, widget, damage );

  PtBasicWidgetCanvas( widget, &canvas ) ;
  PtClipAdd( widget, &canvas ) ;

  print_draw_preview( widget ) ;

  PtClipRemove();
}

//======
// Нарисовать предварительный просмотр.
//
int print_draw_preview( PtWidget_t *widget )
{
  int       line_num = 0, file_line_num = 0 ;
  PtArg_t   args[1] ;
  int       *scroll_pos ;

  if( !print_preview_is_shown) return 0 ;
  if( widget != active_dialog->preview_wgt ) return 0 ;

  // Получить положение скролбара.
  PtSetArg( args, Pt_ARG_SCROLL_POSITION, &scroll_pos, 0 ) ;
  PtGetResources( active_dialog->scrollbar_wgt, 1, args ) ;

  PgSetStrokeColor( preview_color ) ;
  PgSetTextColor( preview_color ) ;
  PgSetFont( preview_font ) ;
```

```
  //*
  //* Отобразить соответствующую часть файла предварительного просмотра.
  //*
  line_num = 0 ;
  file_line_num = -1 ;
  while( fgets( str, 999, print_preview_file ) )
  {
    int       col_num = 0 ;

    file_line_num++ ;

    // Вышли за пределы видимой области - закончить чтение файла.
    if( file_line_num > *scroll_pos + lines_on_screen ) break ;

    // Еще не дошли до видимой области - продолжать считывание.
    if( file_line_num < *scroll_pos ) continue ;

    str[line_width] = 0 ;

    while( str[col_num] != 0 )
    {
      //print_draw_symbol( str[col_num], line_num, col_num ) ;
      A2( 0, str[col_num], line_num, col_num ) ;

      col_num++ ;
    }

    A2( 500, 0, line_num, col_num ) ;

    line_num++ ;
  }

  fseek( print_preview_file, 0, SEEK_SET ) ;

  return 1 ;
}

//======
// Подготовить все для запуска предварительного просмотра.
// Вернуть 0 в случае невозможности завершить подготовку.
//
int print_setup_preview( preview_dialog_t *d )
{
  PtArg_t   args[4] ;
  int       n = 0 ;
  PhDim_t   *dim ;

  //*
  //* Посчитать и откорректировать размер символов для используемого шрифта.
  //*
  PgExtentText( &symbol_size, NULL, preview_font, "A", 1 ) ;

  // "+1" для учета расстояния между соседними символами.
  symbol_size.lr.x = symbol_size.lr.x - symbol_size.ul.x + 1 ;
  symbol_center.x = symbol_size.lr.x / 2 ;

  // "-2" для более плотного расположения строк.
  //symbol_center.y = symbol_size.lr.y ;
  symbol_size.lr.y = symbol_size.lr.y - symbol_size.ul.y - 2 ;
  symbol_center.y = symbol_size.lr.y / 2 ;

  symbol_size.ul.x = 0 ; symbol_size.ul.y = 0 ;

  //*
  //* Посчитать количество строк в файле и размер области просмотра.
  //*
  if( ! (print_preview_file = fopen( print_preview_file_name, "r" )) ) return 0 ;

  lines_in_file = 0 ;
  while( fgets( str, 999, print_preview_file ) )
    lines_in_file++ ;

  fseek( print_preview_file, 0, SEEK_SET ) ;

  PtSetArg( args, Pt_ARG_DIM, &dim, 0 ) ;
  PtGetResources( d->preview_wgt, 1, args ) ;
  lines_on_screen = dim->h / symbol_size.lr.y + 1 ;
```

```
  n = 0 ;
  PtSetArg( &args[n++], Pt_ARG_MAXIMUM, lines_in_file, 0 ) ;
  PtSetArg( &args[n++], Pt_ARG_SCROLL_POSITION, 0, 0 ) ;
  PtSetArg( &args[n++], Pt_ARG_SLIDER_SIZE, lines_on_screen, 0 ) ;
  PtSetArg( &args[n++], Pt_ARG_PAGE_INCREMENT, lines_on_screen-3, 0 ) ;
  PtSetResources( d->scrollbar_wgt, n, args ) ;

  return 1 ;
}
```

## 9.2. Менеджер печати

### 9.2.1. Входные переменные (файл x.c)

```
//=====================================================================
//
// Модуль содержит функции, реализующие входные переменные.
//
//

#include <sys/dev.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include "defines.h"
#include "log.h"

extern char     print_suspend_semaphor_file_name[] ;
extern char     print_spool_file_name[]  ;
extern char     print_current_dir[] ;

extern FILE     *print_spool_file ;
extern int      print_port_desc ;

int x10()
{
  int           result = 0 ;
  struct stat   buf ;

  if( stat( print_spool_file_name, &buf ) == 0 )
    result = buf.st_size == 0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x10 - файл печати имеет нулевую длину", result ) ;
  #endif

  return result ;
} ;

int x50()
{
  int       result = 1 ;

  if( print_spool_file )
  {
    int         read_len = 0, read_buf ;
    long int    pos ;

    pos = ftell( print_spool_file ) ;
    read_len = fread( &read_buf, 1, 1, print_spool_file ) ;
    fseek( print_spool_file, pos, SEEK_SET ) ;

    result = read_len == 0 ;
  }

  #ifdef INPUTS_LOGGING
    log_input( "x50 - конец файла спулинга", result ) ;
  #endif

  return result ;
} ;
```

```
int x60()
{
  int           result = 0 ;
  struct stat   buf ;

  result = stat( print_suspend_semaphor_file_name, &buf ) == 0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x60 - suspend enabled", result ) ;
  #endif

  return result ;
} ;

int x100()
{
  int           result = 1 ;

  result = dev_state( print_port_desc, _DEV_EVENT_OUTPUT, _DEV_EVENT_OUTPUT ) ? 1:0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x100 - порт принтера свободен", result ) ;
  #endif

  return result ;
} ;
```

### 9.2.2. Выходные воздействия (файл z.c)

```
//=====================================================================
//
// Модуль содержит функции, реализующие выходные воздействия.
//
//

#include <sys/kernel.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <share.h>
#include <unistd.h>
#include "log.h"
#include "defines.h"
#include "nls_api.h"

extern char     print_spool_file_name[]  ;
extern FILE     *print_spool_file ;
extern int      print_buffer_size ;
extern int      print_port_desc ;

void z50()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z50. Урезать файл спулинга.", 0 ) ;
  #endif

  {
    int     filedes = -1 ;

    filedes = sopen( print_spool_file_name, O_WRONLY | O_CREAT | O_TRUNC,
                     SH_DENYWR, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP ) ;

    if( filedes != -1 )
      close( filedes ) ;
   }
} ;

void z60()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z60. Напечатать очередной фрагмент файла спулинга.", 0 ) ;
  #endif
```

```
  if( print_spool_file )
  {
    char        buf[1000] = "" ;
    int         read_len = 0 ;
    long int    pos ;

    pos = ftell( print_spool_file ) ;
    read_len = fread( buf, 1, print_buffer_size, print_spool_file ) ;
    fseek( print_spool_file, pos+read_len, SEEK_SET ) ;

    write( print_port_desc, buf, read_len ) ;
    flushall() ;
  } ;
} ;

void z80_0()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z80_0. Закрыть файл спулинга.", 0 ) ;
  #endif

  if( print_spool_file )
  {
    fclose( print_spool_file ) ;
    print_spool_file = NULL ;
  }
} ;

void z80_1()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z80_1. Открыть файл спулинга.", 0 ) ;
  #endif

  print_spool_file = fopen( print_spool_file_name, "r" ) ;
} ;
```

### 9.2.3. Модуль инициализации программы (файл Print_manager.c)

```
#include <sys/sched.h>
#include <sys/stat.h>
#include <sys/dev.h>
#include <sys/proxy.h>
#include <sys/kernel.h>
#include <signal.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include "z.h"
#include "log.h"
#include "A.h"

//*****************************************************************
//* Параметры для настройки менеджера печати.
//*

// Имя порта, к которому подключен принтер.
const char          print_port_name[PATH_MAX] = "/dev/ser2" ;
//const char          print_port_name[PATH_MAX] = "/hd/print.test" ;

// Имя временного файла для печати - файла спулинга (с полным путем).
const char          print_spool_file_name[PATH_MAX] = "/hd/print.spool" ;

const char          print_suspend_semaphor_file_name[PATH_MAX] = "/hd/print.suspend" ;

// Период, с которым будет осуществляться посылка символьных пакетов на принтер (сек).
const int           T70_interval = 1 ;

// Количество символов в пакете
// (значение должно соответствовать быстродействию принтера!).
const int           print_buffer_size = 50 ;

// Дескриптор временного файла для печати.
FILE                *print_spool_file = NULL ;
```

```
// Дескриптор порта, к которому подключен принтер.
int                 print_port_desc = -1 ;

void main()
{
  pid_t                   proxy_pid = -1, client_pid = -1 ;

  // Установить приоритет.
  setprio( 0, 1 ) ;

  // Подключится к порту.
  print_port_desc = open( print_port_name, O_RDWR ) ;
  if( print_port_desc == -1 )
  {
    perror( "Error opening printer port" ) ;
    exit( -1 ) ;
  }

  dev_state( print_port_desc, 0, _DEV_EVENT_OUTPUT ) ;
  dev_osize( print_port_desc, print_buffer_size ) ;

  // Запустить таймер печати.
  {
    timer_t                 timer_id = -1 ;
    struct sigevent         t_event ;
    struct itimerspec       t_value ;

    proxy_pid = qnx_proxy_attach( 0, 0, 0, -1 ) ;
    if( proxy_pid == -1 )
    {
      log_write( '!', "ERROR IN TIMERS!!!", errno ) ;
      exit(-1) ;
    } ;

    t_event.sigev_signo = -proxy_pid ;
    timer_id = timer_create( CLOCK_REALTIME, &t_event ) ;
    if( timer_id == -1 )
    {
      log_write( '!', "ERROR IN TIMERS!!!", errno ) ;
      exit(-1) ;
    } ;

    t_value.it_value.tv_sec = T70_interval ;
    t_value.it_value.tv_nsec = 0 ;
    t_value.it_interval.tv_sec = T70_interval ;
    t_value.it_interval.tv_nsec = 0 ;
    timer_settime( timer_id, 0, &t_value, NULL ) ;
  }

  // Произвести инициализацию.
  z50() ;

  // Основной цикл работы менеджера печати.
  for(;;)
  {
    int     msg ;

    client_pid = Receive( 0, &msg, sizeof( msg )) ;
    Reply( client_pid, NULL, 0 ) ;

    if( client_pid == proxy_pid )
    {
      A1( 70 ) ;
      continue ;
    }
  }
}
```

### 9.2.4. Модуль протоколирования (файл log.c)

```
//=================================================================
//
// Модуль протоколирования работы программы.
//
//

#include <time.h>
#include <sys/timeb.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

#include "log.h"
#include "defines.h"

//======
// Записать строку в протокол.
//
void log_write( int symbol, const char *str, int errornum )
{
  struct timeb      timebuf ;
  char              tod[26] ;
  char              strer[200] = "" ;

  if( errornum )
    sprintf( strer, "(%s)", strerror(errornum) ) ;

  ftime( &timebuf ) ;
  _ctime( &timebuf.time, tod ) ;

  #ifdef SHOW_TIME
    printf( "%.8s.%.3hu%c %s %s\n",
            &tod[11], timebuf.millitm, symbol, str, strer ) ;
  #else
    printf( "%c %s %s\n",
            symbol, str, strer ) ;
  #endif

  flushall() ;
} ;

//======
// Запротоколировать переход автомата.
//
void log_trans( const char * const a_name, int state_from, int state_to )
{
  char  str[100] = "" ;

  sprintf( str, "%s: перешел из состояния %u в состояние %u", a_name,
           state_from, state_to ) ;
  log_write( LOG_GRAPH_TRANS, str, 0 ) ;
}

//======
// Запротоколировать запуск автомата.
//
void log_begin( const char * const a_name, int state, int event )
{
  char  str[100] = "" ;

  sprintf( str, "%s: в состоянии %u запущен с событием e%u", a_name, state, event ) ;
  log_write( LOG_GRAPH_BEGIN, str, 0 ) ;
}

//======
// Запротоколировать завершение автомата.
//
void log_end( const char * const a_name, int state, int event )
{
  char  str[100] = "" ;

  sprintf( str, "%s: завершил обработку события e%u в состоянии %u", a_name, event,
           state ) ;
  log_write( LOG_GRAPH_END, str, 0 ) ;
}
```

```
//======
// Запротоколировать вызов функции выходной переменной.
//
void log_input( const char * const input_name, int result )
{
  char  str[100] = "" ;

  sprintf( str, "%s - вернул %u", input_name, result ) ;
  log_write( LOG_INPUT, str, 0 ) ;
}
```

### 9.2.5. Заголовочный файл настройки режима протоколирования (файл log.c)

```
//=====================================================================
// Заголовочный файл для настройки версии программы.
//
#ifndef _DEFINES_H_INCLUDED_
#define _DEFINES_H_INCLUDED_

//#define SHOW_TIME
//#define PIANO_MODE
//#define FULL_LOG
//#define SHORT_LOG

#ifdef FULL_LOG

#define ACTIONS_LOGGING
#define INPUTS_LOGGING

#define A1_BEGIN_LOGGING
#define A1_END_LOGGING
#define A1_TRANS_LOGGING
#define A1_ERRORS_LOGGING

#endif

#ifdef SHORT_LOG

#define ACTIONS_LOGGING

#define A1_BEGIN_LOGGING
#define A1_END_LOGGING

#endif

#endif//_DEFINES_H_INCLUDED_
```

# 10. ПРОТОКОЛЫ ФУНКЦИОНИРОВАНИЯ ПОДСИСТЕМЫ

## 10.1. Протоколы функционирования модуля печати

### 10.1.1. Диагностирующий (полный) протокол

В случае системы, для которой разрабатывался модуль печати, протоколы являлись единственным эффективным средством отладки, так как размер исполняемого файла системы, в который внедрялся модуль печати, составлял 5,5Мб, а используемый отладчик не мог загружать файлы такого большого объема.

Благодаря использованию протоколов удалось устранить ошибки во взаимодействии системы с модулем печати. Так, например, из приведенного ниже протокола следует (строки со временем "12:58:26"), что в состоянии 2 "Ручной частный" при нажатии кнопки ПЕЧАТЬ автомат A0 вызывается не только с событием e30, но и с событием e40, что некорректно, хотя и не приводит к ошибке в работе модуля.

Указанная ошибка присутствует в файле print_buttons.c и позднее была устранена.

```
12:56:43.668* z51. Урезать файл предварительного просмотра.
12:56:43.708* z55_0. Disable suspending.
12:57:29.045{ A0: в состоянии 0 запущен с событием e40
12:57:29.065> x10 - выбранный для печати файл существует - вернул 1
12:57:29.065T A0: перешел из состояния 0 в состояние 2
12:57:29.085* z800. Показать окно предварительного просмотра.
12:57:29.095* z45. Очистить окно предварительного просмотра.
12:57:29.105* z51. Урезать файл предварительного просмотра.
12:57:29.135* z220. Начать просмотр выбранного файла.
12:57:29.215} A0: завершил обработку события e40 в состоянии 2
12:58:26.911{ A0: в состоянии 2 запущен с событием e40
12:58:26.921} A0: завершил обработку события e40 в состоянии 2
12:58:26.921{ A0: в состоянии 2 запущен с событием e30
12:58:26.931* z200. Добавить выбранный файл в очередь печати.
12:58:26.991} A0: завершил обработку события e30 в состоянии 2
12:58:39.670{ A0: в состоянии 2 запущен с событием e11
12:58:39.680T A0: перешел из состояния 2 в состояние 0
12:58:39.680} A0: завершил обработку события e11 в состоянии 0
12:58:47.829{ A0: в состоянии 0 запущен с событием e30
12:58:47.829* z210. Добавить выбранные протоколы в очередь печати.
12:58:48.009} A0: завершил обработку события e30 в состоянии 0
12:58:59.099{ A0: в состоянии 0 запущен с событием e12
12:58:59.099T A0: перешел из состояния 0 в состояние 1
12:58:59.109* z260. Напечатать заголовок таблицы событий.
12:58:59.149} A0: завершил обработку события e12 в состоянии 1
12:58:59.149{ A0: в состоянии 1 запущен с событием e110
12:58:59.179> x70 - размер файла печати больше допустимого - вернул 0
12:58:59.199* z500_0. Ответ клиенту - сообщение обработано успешно.
12:58:59.209* z250. Напечатать строку с информацией о событии.
12:58:59.219} A0: завершил обработку события e110 в состоянии 1
12:58:59.239{ A0: в состоянии 1 запущен с событием e110
12:58:59.259> x70 - размер файла печати больше допустимого - вернул 0
12:58:59.279* z500_0. Ответ клиенту - сообщение обработано успешно.
12:58:59.289* z250. Напечатать строку с информацией о событии.
12:58:59.299} A0: завершил обработку события e110 в состоянии 1
12:59:07.518{ A0: в состоянии 1 запущен с событием e13
12:59:07.528> x70 - размер файла печати больше допустимого - вернул 0
12:59:07.528T A0: перешел из состояния 1 в состояние 0
12:59:07.538} A0: завершил обработку события e13 в состоянии 0
```

### 10.1.2. Проверяющий (короткий) протокол

```
13:06:59.886* z51. Урезать файл предварительного просмотра.
13:06:59.926* z55_0. Disable suspending.
13:07:06.055{ A0: в состоянии 0 запущен с событием e12
```

```
13:07:06.065* z260. Напечатать заголовок таблицы событий.
13:07:06.125} A0: завершил обработку события e12 в состоянии 1
13:07:06.125{ A0: в состоянии 1 запущен с событием e110
13:07:06.145* z500_0. Ответ клиенту - сообщение обработано успешно.
13:07:06.165* z250. Напечатать строку с информацией о событии.
13:07:06.185} A0: завершил обработку события e110 в состоянии 1
13:07:06.195{ A0: в состоянии 1 запущен с событием e110
13:07:06.215* z500_0. Ответ клиенту - сообщение обработано успешно.
13:07:06.235* z250. Напечатать строку с информацией о событии.
13:07:06.245} A0: завершил обработку события e110 в состоянии 1
13:07:08.805{ A0: в состоянии 1 запущен с событием e13
13:07:08.815} A0: завершил обработку события e13 в состоянии 0
13:07:18.605{ A0: в состоянии 0 запущен с событием e30
13:07:18.605* z210. Добавить выбранные протоколы в очередь печати.
13:07:18.744} A0: завершил обработку события e30 в состоянии 0
13:07:28.864{ A0: в состоянии 0 запущен с событием e40
13:07:28.874* z800. Показать окно предварительного просмотра.
13:07:28.894* z45. Очистить окно предварительного просмотра.
13:07:28.904* z51. Урезать файл предварительного просмотра.
13:07:28.924* z220. Начать просмотр выбранного файла.
13:07:28.954} A0: завершил обработку события e40 в состоянии 2
13:07:38.053{ A0: в состоянии 2 запущен с событием e40
13:07:38.053} A0: завершил обработку события e40 в состоянии 2
13:07:38.063{ A0: в состоянии 2 запущен с событием e30
13:07:38.073* z200. Добавить выбранный файл в очередь печати.
13:07:38.103} A0: завершил обработку события e30 в состоянии 2
13:07:42.643{ A0: в состоянии 2 запущен с событием e12
13:07:42.643* z260. Напечатать заголовок таблицы событий.
13:07:42.693} A0: завершил обработку события e12 в состоянии 1
13:08:04.601{ A0: в состоянии 1 запущен с событием e13
13:08:04.601} A0: завершил обработку события e13 в состоянии 0
13:08:07.561{ A0: в состоянии 0 запущен с событием e11
13:08:07.571} A0: завершил обработку события e11 в состоянии 0
13:08:09.771{ A0: в состоянии 0 запущен с событием e40
13:08:09.771* z800. Показать окно предварительного просмотра.
13:08:09.781* z45. Очистить окно предварительного просмотра.
13:08:09.791* z51. Урезать файл предварительного просмотра.
13:08:09.841* z220. Начать просмотр выбранного файла.
13:08:09.861} A0: завершил обработку события e40 в состоянии 2
13:08:11.501{ A0: в состоянии 2 запущен с событием e11
13:08:11.511} A0: завершил обработку события e11 в состоянии 0
```

## 10.2. Протоколы функционирования менеджера печати

### **10.2.1.** Диагностирующий (полный) протокол

```
13:13:38.548* z50. Урезать файл спулинга.

13:14:20.556{ A1: в состоянии 0 запущен с событием e70
13:14:20.556> x10 - файл печати имеет нулевую длину - вернул 0
13:14:20.606> x60 - suspend enabled - вернул 0
13:14:20.606* z80_1. Открыть файл спулинга.
13:14:20.616T A1: перешел из состояния 0 в состояние 1
13:14:20.626} A1: завершил обработку события e70 в состоянии 1
13:14:21.556{ A1: в состоянии 1 запущен с событием e70
13:14:21.556> x10 - файл печати имеет нулевую длину - вернул 0
13:14:21.566> x50 - конец файла спулинга - вернул 0
13:14:21.576> x60 - suspend enabled - вернул 0
13:14:21.576> x100 - порт принтера свободен - вернул 0
13:14:21.596} A1: завершил обработку события e70 в состоянии 1
13:14:22.555{ A1: в состоянии 1 запущен с событием e70
13:14:22.555> x10 - файл печати имеет нулевую длину - вернул 0
13:14:22.565> x50 - конец файла спулинга - вернул 0
13:14:22.585> x60 - suspend enabled - вернул 0
13:14:22.595> x100 - порт принтера свободен - вернул 1
13:14:22.645* z60. Напечатать очередной фрагмент файла спулинга.
13:14:22.655} A1: завершил обработку события e70 в состоянии 1
13:14:23.555{ A1: в состоянии 1 запущен с событием e70
13:14:23.555> x10 - файл печати имеет нулевую длину - вернул 0
13:14:23.605> x50 - конец файла спулинга - вернул 0
13:14:23.605> x60 - suspend enabled - вернул 0
13:14:23.605> x100 - порт принтера свободен - вернул 1
13:14:23.625* z60. Напечатать очередной фрагмент файла спулинга.
13:14:23.645} A1: завершил обработку события e70 в состоянии 1
13:14:24.555{ A1: в состоянии 1 запущен с событием e70
```

```
13:14:24.555> x10 - файл печати имеет нулевую длину - вернул 0
13:14:24.575> x50 - конец файла спулинга - вернул 0
13:14:24.585> x60 - suspend enabled - вернул 0
13:14:24.595> x100 - порт принтера свободен - вернул 1
13:14:24.605* z60. Напечатать очередной фрагмент файла спулинга.
13:14:24.615} A1: завершил обработку события e70 в состоянии 1
13:14:25.555{ A1: в состоянии 1 запущен с событием e70
13:14:25.555> x10 - файл печати имеет нулевую длину - вернул 0
13:14:25.565> x50 - конец файла спулинга - вернул 0
13:14:25.575> x60 - suspend enabled - вернул 0
13:14:25.585> x100 - порт принтера свободен - вернул 1
13:14:25.595* z60. Напечатать очередной фрагмент файла спулинга.
13:14:25.605} A1: завершил обработку события e70 в состоянии 1
13:14:26.555{ A1: в состоянии 1 запущен с событием e70
13:14:26.555> x10 - файл печати имеет нулевую длину - вернул 0
13:14:26.565> x50 - конец файла спулинга - вернул 0
13:14:26.575> x60 - suspend enabled - вернул 0
13:14:26.585> x100 - порт принтера свободен - вернул 1
13:14:26.595* z60. Напечатать очередной фрагмент файла спулинга.
13:14:26.615} A1: завершил обработку события e70 в состоянии 1
13:14:27.555{ A1: в состоянии 1 запущен с событием e70
13:14:27.555> x10 - файл печати имеет нулевую длину - вернул 0
13:14:27.555> x50 - конец файла спулинга - вернул 0
13:14:27.575> x60 - suspend enabled - вернул 0
13:14:27.585> x100 - порт принтера свободен - вернул 1
13:14:27.595* z60. Напечатать очередной фрагмент файла спулинга.
13:14:27.605} A1: завершил обработку события e70 в состоянии 1
13:14:28.555{ A1: в состоянии 1 запущен с событием e70
13:14:28.555> x10 - файл печати имеет нулевую длину - вернул 0
13:14:28.565> x50 - конец файла спулинга - вернул 0
13:14:28.575> x60 - suspend enabled - вернул 0
13:14:28.625> x100 - порт принтера свободен - вернул 1
13:14:28.625* z60. Напечатать очередной фрагмент файла спулинга.
13:14:28.625} A1: завершил обработку события e70 в состоянии 1
13:14:29.575{ A1: в состоянии 1 запущен с событием e70
13:14:29.585> x10 - файл печати имеет нулевую длину - вернул 0
13:14:29.595> x50 - конец файла спулинга - вернул 1
13:14:29.605T A1: перешел из состояния 1 в состояние 0
13:14:29.615* z80_0. Закрыть файл спулинга.
13:14:29.625* z50. Урезать файл спулинга.
13:14:29.685} A1: завершил обработку события e70 в состоянии 0
13:14:30.555{ A1: в состоянии 0 запущен с событием e70
13:14:30.555> x10 - файл печати имеет нулевую длину - вернул 1
13:14:30.565} A1: завершил обработку события e70 в состоянии 0

13:14:56.553{ A1: в состоянии 0 запущен с событием e70
13:14:56.553> x10 - файл печати имеет нулевую длину - вернул 0
13:14:56.553> x60 - suspend enabled - вернул 0
13:14:56.563* z80_1. Открыть файл спулинга.
13:14:56.593T A1: перешел из состояния 0 в состояние 1
13:14:56.603} A1: завершил обработку события e70 в состоянии 1
13:14:57.553{ A1: в состоянии 1 запущен с событием e70
13:14:57.553> x10 - файл печати имеет нулевую длину - вернул 0
13:14:57.563> x50 - конец файла спулинга - вернул 0
13:14:57.583> x60 - suspend enabled - вернул 0
13:14:57.593> x100 - порт принтера свободен - вернул 1
13:14:57.643* z60. Напечатать очередной фрагмент файла спулинга.
13:14:57.643} A1: завершил обработку события e70 в состоянии 1

13:15:28.551{ A1: в состоянии 1 запущен с событием e70
13:15:28.551> x10 - файл печати имеет нулевую длину - вернул 0
13:15:28.561> x50 - конец файла спулинга - вернул 0
13:15:28.571> x60 - suspend enabled - вернул 1
13:15:28.581T A1: перешел из состояния 1 в состояние 2
13:15:28.591} A1: завершил обработку события e70 в состоянии 2
13:15:29.551{ A1: в состоянии 2 запущен с событием e70
13:15:29.551> x10 - файл печати имеет нулевую длину - вернул 0
13:15:29.601> x60 - suspend enabled - вернул 1
13:15:29.601} A1: завершил обработку события e70 в состоянии 2

13:15:34.551{ A1: в состоянии 2 запущен с событием e70
13:15:34.551> x10 - файл печати имеет нулевую длину - вернул 0
13:15:34.561> x60 - suspend enabled - вернул 0
13:15:34.561T A1: перешел из состояния 2 в состояние 1
13:15:34.571} A1: завершил обработку события e70 в состоянии 1
13:15:35.580{ A1: в состоянии 1 запущен с событием e70
13:15:35.590> x10 - файл печати имеет нулевую длину - вернул 0
```

```
13:15:35.600> x50 - конец файла спулинга - вернул 0
13:15:35.610> x60 - suspend enabled - вернул 0
13:15:35.620> x100 - порт принтера свободен - вернул 1
13:15:35.620* z60. Напечатать очередной фрагмент файла спулинга.
13:15:35.640} A1: завершил обработку события e70 в состоянии 1
13:15:36.550{ A1: в состоянии 1 запущен с событием e70
13:15:36.550> x10 - файл печати имеет нулевую длину - вернул 0
13:15:36.570> x50 - конец файла спулинга - вернул 0
13:15:36.580> x60 - suspend enabled - вернул 0
13:15:36.600> x100 - порт принтера свободен - вернул 1
13:15:36.610* z60. Напечатать очередной фрагмент файла спулинга.
13:15:36.620} A1: завершил обработку события e70 в состоянии 1

13:15:49.550{ A1: в состоянии 2 запущен с событием e70
13:15:49.550> x10 - файл печати имеет нулевую длину - вернул 1
13:15:49.560T A1: перешел из состояния 2 в состояние 0
13:15:49.560* z80_0. Закрыть файл спулинга.
13:15:49.580* z50. Урезать файл спулинга.
13:15:49.599} A1: завершил обработку события e70 в состоянии 0
13:15:50.549{ A1: в состоянии 0 запущен с событием e70
13:15:50.549> x10 - файл печати имеет нулевую длину - вернул 1
13:15:50.549} A1: завершил обработку события e70 в состоянии 0
```

### 10.2.2. Проверяющий (короткий) протокол

```
13:22:17.923* z50. Урезать файл спулинга.

13:23:00.930{ A1: в состоянии 0 запущен с событием e70
13:23:00.950* z80_1. Открыть файл спулинга.
13:23:00.950} A1: завершил обработку события e70 в состоянии 1
13:23:01.930{ A1: в состоянии 1 запущен с событием e70
13:23:01.930} A1: завершил обработку события e70 в состоянии 1
13:23:02.960{ A1: в состоянии 1 запущен с событием e70
13:23:02.960* z60. Напечатать очередной фрагмент файла спулинга.
13:23:02.970} A1: завершил обработку события e70 в состоянии 1

13:23:19.929{ A1: в состоянии 1 запущен с событием e70
13:23:19.929} A1: завершил обработку события e70 в состоянии 2
13:23:20.929{ A1: в состоянии 2 запущен с событием e70
13:23:20.929} A1: завершил обработку события e70 в состоянии 2

13:23:27.928{ A1: в состоянии 2 запущен с событием e70
13:23:27.928} A1: завершил обработку события e70 в состоянии 1
13:23:28.968{ A1: в состоянии 1 запущен с событием e70
13:23:28.978* z60. Напечатать очередной фрагмент файла спулинга.
13:23:28.988} A1: завершил обработку события e70 в состоянии 1
13:23:29.928{ A1: в состоянии 1 запущен с событием e70
13:23:29.928* z60. Напечатать очередной фрагмент файла спулинга.
13:23:29.928} A1: завершил обработку события e70 в состоянии 1

13:23:38.927} A1: завершил обработку события e70 в состоянии 2
13:23:39.927{ A1: в состоянии 2 запущен с событием e70
13:23:39.927} A1: завершил обработку события e70 в состоянии 2
13:23:40.957{ A1: в состоянии 2 запущен с событием e70

13:23:48.927{ A1: в состоянии 2 запущен с событием e70
13:23:48.927* z80_0. Закрыть файл спулинга.
13:23:48.947* z50. Урезать файл спулинга.
13:23:48.977} A1: завершил обработку события e70 в состоянии 0

13:24:25.924{ A1: в состоянии 0 запущен с событием e70
13:24:25.934* z80_1. Открыть файл спулинга.
13:24:25.934} A1: завершил обработку события e70 в состоянии 1
13:24:26.924{ A1: в состоянии 1 запущен с событием e70
13:24:26.924* z60. Напечатать очередной фрагмент файла спулинга.
13:24:26.934} A1: завершил обработку события e70 в состоянии 1
13:24:27.924{ A1: в состоянии 1 запущен с событием e70
13:24:27.934* z60. Напечатать очередной фрагмент файла спулинга.
13:24:27.944} A1: завершил обработку события e70 в состоянии 1

13:24:34.924{ A1: в состоянии 1 запущен с событием e70
13:24:34.924* z80_0. Закрыть файл спулинга.
13:24:34.924* z50. Урезать файл спулинга.
13:24:34.974} A1: завершил обработку события e70 в состоянии 0
13:24:35.923{ A1: в состоянии 0 запущен с событием e70
```