Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Управление ресурсами в вычислительных системах

Лабораторная работа №5

Факультет: прикладной математики и информатики

Группа: ПМ-63

Студенты: Кожекин М.В.

Утюганов Д.С.

Преподаватели: Стасышин В. М.

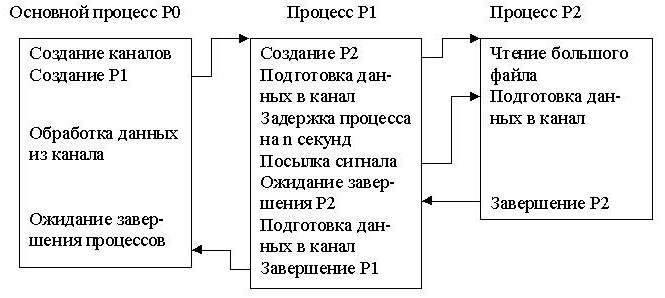
Хайленко Е. А.

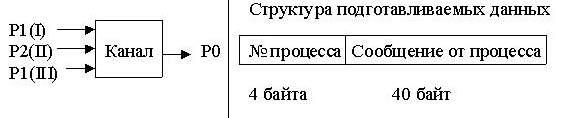
Новосибирск

2019

1. Условие (Вариант №7)

Исходный процесс создает программный канал К1 и порождает новый процесс Р1, а тот, в свою очередь, порождает ещё один процесс Р2. Подготовленные данные последовательно помещаются процессами-сыновьями в программный канал и передаются основному процессу. Файл, читаемый процессом Р2, должен быть достаточно велик с тем, чтобы его чтение не завершилось ранее, чем закончится установленная задержка в n секунд. После срабатывания будильника процесс Р1 посылает сигнал процессу Р2, прерывая чтение файла. Схема взаимодействия процессов, порядок передачи данных в канал и структура подготавливаемых данных показаны ниже:





Обработка данных основным процессом заключается в чтении информации из программного канала и печати её. Кроме того, посредством выдачи сообщений необходимо информировать обо всех этапах работы программы (создание процесса, завершение посылки данных в канал и т.д.).

1. Анализ задачи

Создаём общий канал и порождаем процессы P1 и P2.

Для процессов

**P0:**

Читаем данные из канала и выводим их на экран

Ожидаем завершение процесса P1

**P1:**

Посылаем в канал сообщение, что процесс создан

Делаем задержку на n секунд

Посылаем процессу P2 сигнал прерывания

Сообщаем в канал сообщение о завершении P2

**P2:**

Посылаем в канал сообщение, что процесс создан

Читаем большой файл пока не получим сигнал

1. Используемые программные средства

**Функции:**

int fork() – порождение процесса-потомка

int pipe(int[2]) – порождение канала

int close(const char\*file) – закрытие файла или канала

int sprintf(char \*dest, char \*source1, … , char \*sourcen) -

int kill(pid\_t pid, int sygnal) – передача сигнала

int open(const char\*path, int mode) – открытие файла или именованного канала

int read(int fd, char \*data, int dataSize) – чтение из канала / файла

int wait(int \*status) – ожидание завершения процесса-потомка

int getpid() – определение pid текущего процесса

int sleep(int seconds) – остановка процесса на n секунд

1. Спецификация

Программа находится в папке ~/lab5

Чтобы собрать программу нужно ввести “make lab5

Чтобы запустить программу, нужно использовать команду “./lab5”

В результате работы программы, будет показаны сообщения о действиях процессов P0, P1 и P2 такие как создание / завершение процессов, чтение файла процессом P2 и его прерывание процессом P1.

1. Исходный код

**Файл lab5.c**

|  |
| --- |
| #include <signal.h>  #include <stdio.h>  #include <stdlib.h>  #include <sys/types.h>  #include <unistd.h>  #include <string.h>  #include <sys/ioctl.h>  #include <fcntl.h>  const int BUF\_SIZE = 128;  const int N\_SECONDS = 1;  const int MESSAGES\_COUNT = 4;  int pipeR, pipeW;  // Структура подготавливаемых данных  struct {  int number;  char message[128];  } data;  // Отправка данных основному процессу P0  void sendToPipe(int number, char \*message) {  if (strlen(message)+1>128)  {  printf("ОШИБКА. Сообщение больше 128 байт");  exit(-1);  }  data.number = number;  strcpy(data.message, message);  write(pipeW, &data, sizeof(data));  }  int main()  {  char string[128];  int fd[2];  if (pipe(fd) == -1)  {  printf("Ошибка создания канала\n");  exit(-1);  }  pipeW = fd[1];  pipeR = fd[0];  pid\_t pidP0 = getpid();  pid\_t pidP1 = fork();  pid\_t pidP2 = 0;  pid\_t pidP;  printf("P0 (%d):\tСоздание каналов чтения и записи\n", pidP0);  int returnStatus;  if (pidP1 == 0) {  close(pipeR);  pidP = 1;  pidP1 = getpid();  pidP2 = fork();  if (pidP2 == 0) {  pidP = 2;  pidP2 = getpid();  }  }  else{  close(pipeW);  pidP = 0;  }  switch (pidP)  {  // Процесс P0  case 0: {  printf("P0 (%d):\tСоздание P1\n", pidP0);  printf("P0 (%d):\tОбработка данных из канала\n", pidP0);  for (int i = 0; i < MESSAGES\_COUNT; i++)  {  read(pipeR, &data, sizeof(data));  printf("P0: %s\n", data.message);  }  printf("P0 (%d):\tОжидание завершения процессов\n", pidP0);  wait(&returnStatus);  printf("P0 (%d):\tПроцесс P1 завершился со статусом %d\n", pidP0, returnStatus);  }break;  // Процесс P1  case 1: {  close(pipeR);  sprintf(string,"P1 (%d):\tСоздание P2", pidP1);  sendToPipe(1,string);  sleep(N\_SECONDS);  // Прерываем процесс P2  kill(pidP2, SIGINT);  wait(&returnStatus);  sprintf(string,"P1 (%d):\tПроцесс P2 завершился со статусом %d", pidP1, returnStatus);  sendToPipe(1,string);  }break;  // Процесс P2  case 2: {  sprintf(string, "P2 (%d):\tОткрытие большого файла", pidP2);  sendToPipe(1,string);  FILE \*file\_in = fopen("input.txt", "r");  if (file\_in == NULL)  {  sprintf(string,"P2 (%d):\tОшибка открытия файла", pidP2);  sendToPipe(1,string);  return -1;  }  else  {  sprintf(string, "P2 (%d):\tЧтение большого файла", pidP2);  sendToPipe(1,string);  char \*string = NULL;  while (fgets(string, BUF\_SIZE, file\_in) != -1);  return 0;  }  }break;  default:  printf("Ошибка процесса\n");  break;  }  } |

**Содержимое файла Makefile**

|  |
| --- |
| # Makefile for lab #5  lab5:  gcc lab5.c -std=gnu99 -o lab5  clean:  rm lab5  run:  rm lab5 lab5.c; nano lab5.c; gcc lab5.c -std=gnu99 -o lab5; ./lab5 |

1. Результат работы программы

Программа работает должным образом

|  |
| --- |
| -bash-4.2$ ./lab5  P0 (15814): Создание каналов чтения и записи  P0 (15814): Создание P1  P0 (15814): Обработка данных из канала  P0 (15814): Создание каналов чтения и записи  P0: P1 (15823): Создание P2  P0: P2 (15832): Открытие большого файла  P0: P2 (15832): Чтение большого файла  P0: P1 (15823): Процесс P2 завершился со статусом 139  P0 (15814): Ожидание завершения процессов  P0 (15814): Процесс P1 завершился со статусом 0 |