Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Управление ресурсами в вычислительных системах

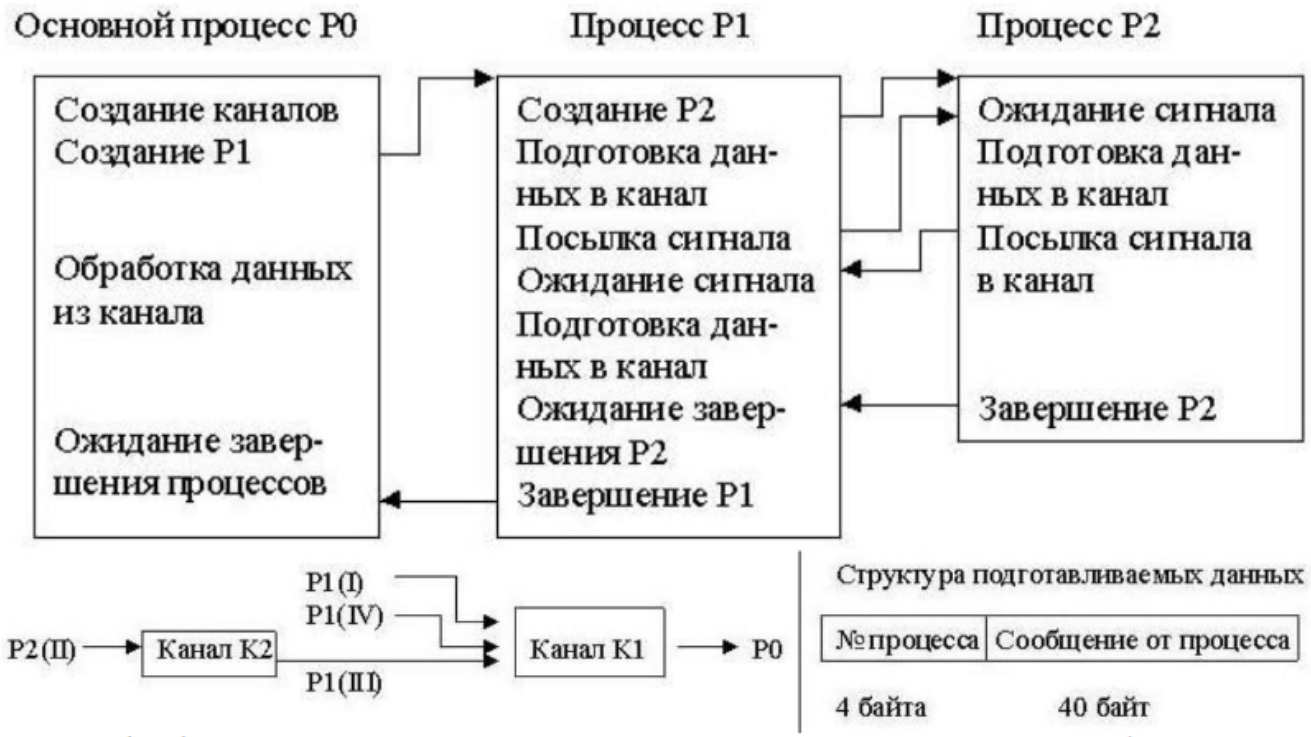
Лабораторная работа №5

Группа:	ПМ-13
Студенты:	Исакин Д.А.
	Вострецова Е.В.
Преподаватели:	Стасышин В.М.
	Сивак М.А.

Новосибирск
2024

1. Условие (Вариант №1)

Исходный процесс создает два программных канала К1 и К2 и порождает новый процесс Р1, а тот, в свою очередь, еще один процесс Р2, каждый из которых готовит данные для обработки их основным процессом. Подготавливаемые данные процесс Р1 помещает в канал К1, а процесс Р2 в канал К2, откуда они процессом Р1 копируются в канал К1 и дополняются новой порцией данных. 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 К1 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.).

2.Анализ задачи

Создаём каналы K1 и K2 и порождаем процесс P1 и в процессе P1 пораждаем дочерний процесс P2
Для процессов
P0:

1 Читаем данные из канала и выводим их на экран. Чтение производим по сигналу от дочернео процесса о том, что данные готовы

2 Ожидаем завершение процесса P1

P1:

1 Создаем процесс P2

2 Готовим данные от P1 и отправляем их в K1. После посылаем сигнал P0

3 Ждем от P2 сигнала готовности

4 Читаем данные из K2 и пишем их в K1. Посылаеп сигнал P0

5 Модификация данных. Пишем в K1. Посылаем P0 сигнал готовности. После сигнал завершения

P2:

1 Готовим данные

2 Пишем в K2

3.Используемые программные средства.

```
int fork(); - порождение процессапотомка-
int pipe(int[2]); - порождение канала
int fclose(const char*file); - закрытие файла или канала
int kill(pid_t pid, int sygnal); - передача сигнала
int read(int fd, char *data, int dataSize); - чтение из канала файла
int waitpid(pid_t pid,int *status, NULL); - ожидание завершения процессапотомка-
int getpid(); - определение pid текущего процесса
int getppid(); - определение процесса родителя
int sleep(int seconds); - остановка процесса на n секунд
```

4. Спецификация

- Программа находится в папке /lab3
- Чтобы собрать программу нужно ввести make
- Чтобы запустить программу, нужно использовать команду "./lab3"
- В результате работы программы, будет показаны сообщения о действиях процессов P0, P1 и P2 такие как создание / завершение процессов, чтение файла процессом P0. Завершение процессов потомков.

5. Результат работы программы

```
P0: Try to create K1
P0: K1 create success
P0: Try to create K2
P0: K2 create success

P0: Try to create P1
P1: P1 create sucess. pid(P1) = 0
P1: Try to create P2
P1(1): Data wreaten and send to K1
P0: pid = 148244, data = Data: P1
P2: P2 create sucess. pid(P2) = 0
P2(2): Data writen and sended to K2
P1: Process P2 was end correct
P1(3): Data from P2 got and written to K1 and sended
P1(4): Data P1 modifie and send to K1
P0: break;
P0: Process P1 was end correct
```

6. Исходный код

main.c

```c
#include <stdio.h>      // io-functions
#include <sys/types.h> // pid_t
#include <unistd.h>     // fork(), sleep(), usleep()
#include <stdlib.h>     // fprintf(), fscanf()
#include <string.h>
#include <inttypes.h>
#include <sys/wait.h>
#include <sys/signal.h>

#define SIGBREAK SIGUSR2 // Завершает P0

int state = 0;

struct _Data
{
    pid_t pid;
    char data[40];
};

void handler(int sig)
{
    if (sig == SIGCONT)
    {
        // printf("signal set 1\n");
        state = 1; // Процесс готов начать обработку данных
    }
    else if (sig == SIGBREAK)
    {
        state = 3; //  Команда завершения цикла
    }
}

typedef struct _Data Data;

int main()
{

    /* Регистрация обработчиков сигналов */
    if((signal(SIGCONT, handler)) == SIG_ERR)
    {
        fprintf(stderr, "Ошибка назначения сигнала\n");
        exit(EXIT_FAILURE);
    }

    if((signal(SIGBREAK, handler)) == SIG_ERR)
    {
        fprintf(stderr, "Ошибка назначения сигнала\n");
        exit(EXIT_FAILURE);
    }

    pid_t P1; // ID 1 ого процесса
    pid_t P2; // ID 2 ого процесса порожденного в ом1-

    int32_t K1[2]; // Програмный канал номер 1; K1[0] - дескриптор для чтения, K1[1] - дескриптор для записи
    int32_t K2[2]; // Програмный канал номер 2; Аналогично K1

    /* Инициализация програмныйх каналов */
    printf("P0: Try to create K1\n");
    if (pipe(K1))
```

```
60      {
61          fprintf(stderr, 'P0: Pipe failed.\n');
62          return EXIT_FAILURE;
63      }
64      printf("P0: K1 create success\n");
65
66      printf("P0: Try to create K2\n");
67      if (pipe(K2))
68      {
69          fprintf(stderr, 'P0: Pipe failed.\n');
70          return EXIT_FAILURE;
71      }
72      printf("P0: K2 create success\n\n");
73
74      /************************************/
75
76      // Создание процесса P1
77      printf("P0: Try to create P1\n");
78      if ((P1 = fork()) == 0)
79      {
80          int killStatus = 0;
81
82          usleep(100); // тактирование
83
84          printf("P1: P1 create sucess. pid(P1) = %d\n", P1);
85
86          printf("P1: Try to create P2\n");
87          if ((P2 = fork()) == 0)
88          {
89              printf("P2: P2 create sucess. pid(P2) = %d\n", P2);
90              // Подготовка данных
91              Data dataP2 = {getpid(), "Data: P2"};
92              write(K2[1], &dataP2.pid, 4);
93              write(K2[1], dataP2.data, 40);
94              printf("P2(2): Data writen and sended to K2\n");
95              exit(EXIT_SUCCESS);
96          }
97          else if (P2 < (pid_t)0)
98          {
99              /* The fork failed. */
100             fprintf(stderr, 'P1: Fork failed.\n');
101             exit(EXIT_FAILURE);
102         }
103
104         // Подготовка данных
105         Data dataP1 = {getpid(), "Data: P1"}; // Подготовленные данные для канала
    K1 от P1
106
107         // Пишем данные в канал K1 P1(1)
108         write(K1[1], &dataP1.pid, 4);
109         write(K1[1], dataP1.data, 40);
110         printf("P1(1): Data wreaten and send to K1\n");
111         while(kill(getppid(), SIGCONT)); // state = 1; Сигнал отправки данных
112
113         //usleep(100);                // Тактирование
114         /************************************************************/
115
116         /* Ждем корректного завершения P2 */
117         int statusP2;
118         waitpid(P2, &statusP2, NULL);
119         if (statusP2 == 0)
120         {
121             printf("P1: Process P2 was end correct\n");
```

```
122         }
123         else
124         {
125             fprintf(stderr, "P1: P2 error status = %d\n", statusP2);
126             return EXIT_FAILURE;
127         }
128         /* Уверенны, что данные в K2 уже есть читаем их и сохраняем */
129         Data dataP2;
130         read(K2[0], &dataP2.pid, 4);
131         read(K2[0], dataP2.data, 40);
132         // Пишем эту порцию данных в K1 и посылаем сигнал P0 на прием
133         write(K1[1], &dataP2.pid, 4);
134         write(K1[1], dataP2.data, 40);
135         printf("P1(3): Data from P2 got and written to K1 and sended\n");
136         while(kill(getppid(), SIGCONT));
137         //usleep(100); // Тактирование
138 
139         /* Формируем 4 набор данных для P0 */
140         strcat(dataP1.data, " ");
141         strcat(dataP1.data, dataP2.data);
142         // Пишем данные в канал K1 P1(1)
143         write(K1[1], &dataP1.pid, 4);
144         write(K1[1], dataP1.data, 40);
145         printf("P1(4): Data P1 modifie and send to K1\n");
146         while(kill(getppid(), SIGCONT)); // state = 1; Сигнал отправки данных
147         //usleep(100);               // Тактирование
148         /* Выход из цикла в P0 - можно было это на waitpid повесить, но так хотя бы
    сигналы поиспользовали */
149         usleep(100); // Тактируем
150         while(kill(getppid(), SIGBREAK));
151 
152         exit(EXIT_SUCCESS);
153     }
154     // Не удалось создать дочерний процесс
155     else if (P1 < (pid_t)0)
156     {
157         /* The fork failed. */
158         fprintf(stderr, 'P0: Fork failed.\n');
159         exit(EXIT_FAILURE);
160     }
161     // Родитель
162     else
163     {
164         int32_t status;
165         /* Ждем сигналов от потомка и читаем данные из канала K1 */
166         /* Обработчик для чтения */
167 
168         while (1)
169         {
170             if (state == 1)
171             {
172                 Data data;
173                 read(K1[0], &data.pid, 4);
174                 read(K1[0], data.data, 40);
175                 printf("P0: pid = %d, data = %s\n", data.pid, data.data);
176                 state = 0;
177             }
178             else if (state == 3)
179             {
180                 printf("P0: break;\n");
181                 break;
182             }
183             usleep(100); // Ждем
```

```
184         }
185 
186         waitpid(P1, &status, NULL); // Ждем P1, когда он завершится читаем данные
    из канала K1
187         if (status == 0)
188         {
189             printf("P0: Process P1 was end correct\n");
190         }
191         else
192         {
193             fprintf(stderr, "P0: P1 error status = %d\n", status);
194             return EXIT_FAILURE;
195         }
196 
197         // printf("wait P1, pid(P1) = %d\n", P1);
198     }
199 
200     return EXIT_SUCCESS;
201 }
```

makefile

```
1  # Makefile for lab #2
2  all: main
3  
4  main: main.o
5      gcc -std=c11 main.o -o main
6  
7  main.o: main.c
8      gcc -std=c11 -c main.c
9  
10 clean:
11     rm -rf *.o main
```