

Управление ресурсами в вычислительных системах

ЛАБОРАТОРНАЯ РАБОТА №3

Группа:

ПМ-13

Студенты:

Исакин Д.А.

Вострецова Е.В.

Преподаватели:

Стасышин В.М.

Сивак М.А.

1. Условие (Вариант №1)

Исходный процесс создает два программных канала K1 и K2 и порождает новый процесс P1, а тот, в свою очередь, еще один процесс P2, каждый из которых готовит данные для обработки их основным процессом. Подготавливаемые данные процесс P1 помещает в канал K1, а процесс P2 в канал K2, откуда они процессом P1 копируются в канал K1 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 K1 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 порождаем 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. Посылаем 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.

```
1 int fork(); - порождение процесспотомка-
2 int pipe(int[2]); - порождение канала
3 int fclose(const char*file); - закрытие файла или канала
4 int kill(pid_t pid, int signal); - передача сигнала
5 int read(int fd, char *data, int dataSize); - чтение из канала файла
6 int waitpid(pid_t pid,int *status, NULL); - ожидание завершения процесспотомка-
7 int getpid(); - определение pid текущего процесса
8 int getppid(); - определение процесса родителя
9 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

```
1 P0: Try to create K1
2 P0: K1 create success
3 P0: Try to create K2
4 P0: K2 create success
5
6 P0: Try to create P1
7 P1: P1 create sucess. pid(P1) = 0
8 P1: Try to create P2
9 P1(1): Data wreaten and send to K1
10 P0: pid = 148244, data = Data: P1
11 P2: P2 create sucess. pid(P2) = 0
12 P2(2): Data writen and sended to K2
13 P1: Process P2 was end correct
14 P1(3): Data from P2 got and written to K1 and sended
15 P1(4): Data P1 modifie and send to K1
16 P0: break;
17 P0: Process P1 was end correct
```

6. Исходный код

main.c

```
1 #include <stdio.h>           // io-functions
2 #include <sys/types.h>       // pid_t
3 #include <unistd.h>          // fork(), sleep(), usleep()
4 #include <stdlib.h>          // fprintf(), fscanf()
5 #include <string.h>
6 #include <inttypes.h>
7 #include <sys/wait.h>
8 #include <sys/signal.h>
9
10 #define SIGBREAK SIGUSR2 // Завершает P0
11
12 int state = 0;
13
14 struct _Data
15 {
16     pid_t pid;
17     char data[40];
18 };
19
20 void handler(int sig)
21 {
22     if (sig == SIGCONT)
23     {
24         // printf("signal set 1\n");
25         state = 1; // Процесс готов начать обработку данных
26     }
27     else if (sig == SIGBREAK)
28     {
29         state = 3; // Команда завершения цикла
30     }
31 }
32
33 typedef struct _Data Data;
34
35 int main()
36 {
37
38     /* Регистрация обработчиков сигналов */
39     signal(SIGCONT, handler);
40     signal(SIGBREAK, handler);
41
42     pid_t P1; // ID 1 ого процесса
43     pid_t P2; // ID 2 ого процесса порожденного в om1-
44
45     int32_t K1[2]; // Програмный канал номер 1; K1[0] - дескриптор для чтения, K1[1]
- дескриптор для записи
46     int32_t K2[2]; // Програмный канал номер 2; Аналогично K1
47
48     /* Инициализация программных каналов */
49     printf("P0: Try to create K1\n");
50     if (pipe(K1))
51     {
52         fprintf(stderr, 'P0: Pipe failed.\n');
53         return EXIT_FAILURE;
54     }
55     printf("P0: K1 create success\n");
56
57     printf("P0: Try to create K2\n");
58     if (pipe(K2))
59     {
```

```

60     fprintf(stderr, 'P0: Pipe failed.\n');
61     return EXIT_FAILURE;
62 }
63 printf("P0: K2 create success\n\n");
64
65 /*****/
66
67 // Создание процесса P1
68 printf("P0: Try to create P1\n");
69 if ((P1 = fork()) == 0)
70 {
71     usleep(1000); // тактирование
72
73     printf("P1: P1 create sucess. pid(P1) = %d\n", P1);
74
75     printf("P1: Try to create P2\n");
76     if ((P2 = fork()) == 0)
77     {
78         printf("P2: P2 create sucess. pid(P2) = %d\n", P2);
79         // Подготовка данных
80         Data dataP2 = {getpid(), "Data: P2"};
81         write(K2[1], &dataP2.pid, 4);
82         write(K2[1], dataP2.data, 40);
83         printf("P2(2): Data writen and sended to K2\n");
84         exit(EXIT_SUCCESS);
85     }
86     else if (P2 < (pid_t)0)
87     {
88         /* The fork failed. */
89         fprintf(stderr, 'P1: Fork failed.\n');
90         exit(EXIT_FAILURE);
91     }
92
93     // Подготовка данных
94     Data dataP1 = {getpid(), "Data: P1"}; // Подготовленные данные для канала
K1 от P1
95
96     // Пишем данные в канал K1 P1(1)
97     write(K1[1], &dataP1.pid, 4);
98     write(K1[1], dataP1.data, 40);
99     printf("P1(1): Data wreaten and send to K1\n");
100     kill(getppid(), SIGCONT); // state = 1; Сигнал отправки данных
101     usleep(100); // Тактирование
102     /*****/
103
104     /* Ждем корректного завершения P2 */
105     int statusP2;
106     waitpid(P2, &statusP2, NULL);
107     if (statusP2 == 0)
108     {
109         printf("P1: Process P2 was end correct\n");
110     }
111     else
112     {
113         fprintf(stderr, "P1: P2 error status = %d\n", statusP2);
114         return EXIT_FAILURE;
115     }
116     /* Уверенны, что данные в K2 уже есть читаем их и сохраняем */
117     Data dataP2;
118     read(K2[0], &dataP2.pid, 4);
119     read(K2[0], dataP2.data, 40);
120     // Пишем эту порцию данных в K1 и посылаем сигнал P0 на прием
121     write(K1[1], &dataP2.pid, 4);

```

```

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

```

```
183
184         // printf("wait P1, pid(P1) = %d\n", P1);
185     }
186
187     return EXIT_SUCCESS;
188 }
```

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
```
