Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Операционные системы»

Студент: А. О. Дубинин
Преподаватель: Е. С. Миронов
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №4

Цель работы: Приобретение практических навыков в:
• Освоение принципов работы с файловыми системами
• Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание: Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант 22 :

Родительский процесс представляет собой сервер по работе с массивами и принимает команды со стороны дочернего процесса.

1 Описание

Будем использовать использовать отображение в память mmmap для передачи команд по работе с массивом, синхронизируя действия сигналами.

Системные вызовы, которые были использованы:

pid_t fork(void);	Создает дочерний процесс. Если возвращает 0, то созданный про- цесс – ребенок, если > 0, то – родитель.
pid_t waitpid(pidt pid, int *status, int options);	Приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до по- явления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.
void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset);	Функция mmap отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией mmap, и никогда не бывает равным 0.
int sigemptyset(sigset_t *set);	sigemptyset инициализирует набор сигналов, указанный в set, и "очищает"его от всех сигналов.
int sigaddset(sigset_t *set, int signum);	sigaddse добавляет сигналы signum к set и удаляет эти сигналы из набора соответственно.
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);	Системный вызов sigprocmask используется для того, чтобы изменить список блокированных в данный момент сигналов.

int sigwait(const sigset_t *set, int *sig);	Функция sigwait() приостанавливает выполнение вызвавшей нити до тех пор, пока в ожидании не появится сигнал из заданного набора сигналов set. Функция принимает сигнал (удаляет его из списка ожидающих сигналов) и возвращает номер сигнала в sig.
int kill(pid_t pid, int sig);	Системный вызов kill может быть использован для посылки какого-либо сигнала какому-либо процессу или группе процесса.

2 Исходный код

```
1  //main.c
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <sys/types.h>
6  #include <unistd.h>
7  #include <string.h>
8  #include <sys/wait.h>
9  #include <stdbool.h>
10 #include <sys/mman.h>
11
12
13 #include "vector.h"
14 #include "vector.cpp"
15
16 char read_command() {
17     char command;
18     do {
19         command = getchar();
20     } while ( command == '\n' || command == ' ' );
21     return command;
22 }
23
24 void read_string(char* area) {
25     char c;
26     do {
27         c = getchar();
28     }
29     while(c == ' ' || c == '\n');
30     int size_app_elem = 1;
31
32     while ( true ) {
33
34         area[size_app_elem] = c;
35         size_app_elem++;
36         c = getchar();
37         if ( c == '\n' || c == ' ' ) break;
38
39     }
40     area[size_app_elem] = '\0';
41 }
42
43 void help_function() {
44     printf( "command:\n" );
45     printf( "<h>\thelp\n" );
46     printf( "<q>\texit\n" );
47     printf( "<a>\tappend element into array\n" );
```

```
48      printf( "<p>\tprint array\n" );
49      printf( "<d> <index>\tdelete element in array by index\n" );
50  }
51
52  int main( int argc, char* argv[] ) {
53      int i, id, size;
54      char * element;
55
56      char *area;
57      if ((area = mmap (0, sizeof(char) * 100, PROT_READ | PROT_WRITE, MAP_SHARED |
            MAP_ANONYMOUS, -1, 0))
58          == MAP_FAILED) {
59          printf ("mmap error for input");
60          return 0;
61      }
62
63      //sig set
64      int sig;
65      pid_t parent_id;
66      sigset_t set;
67      sigemptyset(&set);
68      sigaddset(&set, SIGUSR1); // USR1
69      sigaddset(&set, SIGUSR2); // USR2
70      sigprocmask(SIG_BLOCK, &set, NULL);
71      parent_id = getpid(); // id  -
72
73
74
75
76      // create a pipe
77      int my_pipe[2];
78      if ( pipe( my_pipe ) == -1 ) {
79          perror( "Error creating pipe\n" );
80      }
81
82      // fork
83      pid_t child_id;
84      child_id = fork();
85      if ( child_id == -1 ) {
86          perror( "Fork error\n" );
87      }
88
89      if ( child_id == 0 ) // child process
90      {
91
92          if ( close( my_pipe[0] ) < 0 ) { // child doesn't read
93              perror( "Failed to close pipe descriptors" );
94          }
95
```

```
96          char command;
97          printf( "Write a command(<h> for help):\n" );
98          while ( true ) {
99              command = read_command();
100
101             if ( command == EOF ) {
102                 command = 'q';
103                 write( my_pipe[1], &command, 1 );
104                 kill(parent_id, SIGUSR1);
105                 break;
106             }
107
108             area[0] = command;
109
110             switch ( command ) {
111                 case 'a':
112                     printf( "write element:\n" );
113                     read_string(area);
114                     break;
115                 case 'd':
116
117                     read_string(area);
118                     break;
119                 case 'q':
120                     if ( close( my_pipe[1] ) < 0 ) {
121                         perror( "Failed to close pipe descriptors" );
122                     }
123                     kill(parent_id, SIGUSR1);
124                     return 0;
125                 case 'p':
126                 case 'h':
127                     break;
128                 default:
129                     printf( "wrong command\n" );
130                     break;
131
132             }
133             kill(parent_id, SIGUSR1);
134             sigwait(&set, &sig);
135         }
136
137
138     } else // parent process
139     {
140
141
142         vector v;
143         vector_init( &v );
144
```

```
145         if ( close( my_pipe[1] ) < 0 ) { // parent doesn't write
146             perror( "Failed to close pipe descriptors" );
147         }
148 
149         while ( true ) {
150             sigwait(&set, &sig);
151 
152             if ( area[0] == EOF ) {
153                 break;
154             }
155 
156             switch ( area[0] ) {
157                 case 'a':
158                     size = strlen(area);
159                     element = ( char* ) malloc( sizeof( char ) * size );
160                     for ( i = 1; i < size; ++i ) {
161                         element[i-1] = area[i];
162                     }
163                     element[i] = '\0';
164                     vector_add( &v, element );
165                     break;
166                 case 'd':
167                     size = strlen(area);
168                     id = area[1] - '0';
169                     for ( i = 2; i < size; ++i ) {
170                         id = (id * 10) + (area[i] - '0');
171                     }
172                     //printf("%d\n", id);
173                     vector_delete( &v, id );
174                     break;
175                 case 'p':
176                     printf( "vector:\n" );
177                     for ( i = 0; i < vector_count( &v ); i++ ) {
178                         element = vector_get( &v, i );
179                         printf( "%s\n", element );
180                     }
181                     break;
182                 case 'q':
183                     vector_free( &v );
184                     if ( close( my_pipe[0] ) < 0 ) {
185                         perror( "Failed to close pipe descriptors" );
186                     }
187                     return 0;
188                 case 'h':
189                     help_function();
190                     break;
191                 default:
192                     printf( "wrong command\n" );
193                     break;
```

```
194
195             }
196             printf( "Write a command(<h> for help):\n" );
197             kill(child_id, SIGUSR2);
198         }
199
200         if ( close( my_pipe[0] ) < 0 ) {
201             perror( "Failed to close pipe descriptors" );
202         }
203
204
205     }
206
207     return 0;
208 }
```

3 Тесты

```
art@mars:~/workdir/OS/lab_4/cmake-build-debug$ ./variant_22

Write a command(<h>for help):
h
command:
<h>help
<q>exit
<a>append element into array
<p>print array
<d><index>delete element in array by index
Write a command(<h>for help):
a
write element:
1
Write a command(<h>for help):
p
vector:
1
Write a command(<h>for help):
a
write element:
2
Write a command(<h>for help):
p
vector:
1
2
Write a command(<h>for help):
d
0
Write a command(<h>for help):
p
vector:
2
Write a command(<h>for help):
q
```

4 Диагностика

```
art@mars:~/workdir/OS/lab_4/cmake-build-debug$ strace ./variant_22
...
//отображение файла
mmap(NULL,100,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANONYMOUS,-1,0) = 0x7f376d2ee000
rt_sigprocmask(SIG_BLOCK,[USR1 USR2],NULL,8) = 0
getpid()                                = 14762
pipe([3,4])                            = 0
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_tid
= 14763
close(4)                                = 0
Write a command(<h>for help):
rt_sigtimedwait([USR1 USR2],q
si_signo=SIGUSR1,si_code=SI_USER,si_pid=14763,si_uid=1000,NULL,8) = 10 (SIGUSR1)
close(3)                                = 0
---SIGCHLD si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=14763,si_uid=1000,si_status=0,si
---
exit_group(0)                           = ?
+++ exited with 0 +++
```

5 Выводы

Выполнив лабораторную работу, я приобрёл практические навыки работы с файловыми системами. Мною была изучена и применена концепция отображения файла в память процесса file mapping. File mapping может быть полезен когда необходим доступ к случайным областям очень большого файла, выигрыш происходит за счёт отсутствия последовательного считывания. Так же я понял, что проблема гонки ресурсов возникает и здесь, общую память, нужно синхронизировать при попытки обращения к ней.