Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Операционные системы»

Освоение принципов работы с файловыми системами.

Обеспечение обмена данных между процессами посредством технологии «File mapping».

Студент: П. А. Мохляков
Преподаватель: Е. С. Миронов
Группа: М8О-208Б-19
Вариант: 1
Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

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 решения 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.

Родительский процесс передает команды пользователя дочернему процессу. Дочерний процесс принеобходимости передает данные в родительский процесс. Результаты своей работы дочерний процесс пишет в созданный им файл.

Пользователь вводит команды вида: « число число число<endline> ». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int.

2 Сведения о программе

Программа написанна на Си в Unix подобной операционной системе на базе ядра Linux. При компиляции следует линковать библиотеки -lpthread и -lrt. В программе создается дочерний процесс, данные в который передаются с помощью shared memory.

Дочерний прочесс принимает строку чисел и находит их сумму, ответ записывая в файл. Имя файла задается пользователем

Родительский процесс считывает вводные данные у пользовотеля и пердет их дочернему процессу через отброженный участок памати shared memory.

Программа завершает работу при окончании ввода, то есть нажатии CTRL+D.

3 Общий метод и алгоритм решения

Сначала в родительском процессе мы создаем shared memory object и получаем их дискрипторы. Первый файл будет отвечать за сами данные, второй за их размер, а третий за mutex. Далее мы проецируем файлы на память, и инициазируем. Создав перед этим атрибуты для мютекса, в частности для того чтобы он работал для разных процессов.

Далее мы делаем fork() и запускаем в ребенке его программу, передав как оргументы имя результирующего файла, и имена рбъектов shared memory. В дочернем процессе

мы также открываем shared memory и проецируем их.

Потом идет логика самой программы. В начале родитель блокирует mutex, считывает счисло и символ за ним, добавляя число в спроецировннй массив и увличивая счетчик его длины. Если символ за числом будет равен символу конца строки, то мы разблокируем mutex и ожидаем пока переменная длины массива не станет равна нулю, далее блокируем mutex и повторяем цикл. При окончании ввода родитель присваивает переменной длине массива значение -1 и разблокируем mutex.

Ребенок в это время также пытается заблокировать mutex, если у него это получается до ввода родителя, то он его сразу разблокирует и повторяет цикл. Если же ввод был совершен и переменная длины массива больше нуля, то мы считываем Size елементов в массиве и суммируем их, записывая ответ в файл, разблокируем mutex и повторяем цикл. В случае если длина массива данных станет -1, то мы выходим из цикла.

Далее мы в обоих программах отключаем проекции и закрываем файлы.

4 Листинг программы

parent.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h>
4  #include <sys/mman.h>
5  #include <sys/types.h>
6  #include <fcntl.h>
7  #include <pthread.h>
8
9  #define SH_NAME "my_shared_mem"
10 #define SH_SIZE_NAME "my_shared_mem_size"
11 #define MUTEX_NAME "my_mutex"
12
13 void wait(int *elem, int num){
14     while (*elem != num){
15     }
16 }
17
18 int main()
19 {
20     int fd_shared_data = shm_open(SH_NAME, O_RDWR | O_CREAT, S_IRWXU);
21     int fd_shared_data_size = shm_open(SH_SIZE_NAME, O_RDWR | O_CREAT, S_IRWXU);
22     int fd_mutex = shm_open(MUTEX_NAME,O_RDWR | O_CREAT, S_IRWXU);
23
24     if(fd_shared_data == -1 || fd_shared_data_size == -1 || fd_mutex == -1){
25         printf("Error: shared memory open\n");
26         return -1;
27     }
```

```
28      if(ftruncate(fd_shared_data,getpagesize()) == -1){
29          printf("Error: ftruncate\n");
30          return -1;
31      }
32      if(ftruncate(fd_shared_data_size,sizeof(int)) == -1){
33          printf("Error: ftruncate\n");
34          return -1;
35      }
36      if(ftruncate(fd_mutex,sizeof(pthread_mutex_t*)) == -1){
37          printf("Error: ftruncate\n");
38          return -1;
39      }
40
41      int *Data = (int*) mmap(NULL,getpagesize(),PROT_READ | PROT_WRITE, MAP_SHARED,
            fd_shared_data, 0);
42      int *Size = (int*) mmap(NULL,sizeof(int),PROT_READ | PROT_WRITE, MAP_SHARED,
            fd_shared_data_size, 0);
43      pthread_mutex_t *Lock = (pthread_mutex_t*) mmap(NULL,sizeof(pthread_mutex_t*),
            PROT_READ | PROT_WRITE, MAP_SHARED,fd_mutex,0);
44      if (Data == MAP_FAILED || Size == MAP_FAILED || Lock == MAP_FAILED) {
45          printf("Error: map file\n");
46          return -1;
47      }
48      pthread_mutexattr_t MutexAttribute;
49      if(pthread_mutexattr_setpshared(&MutexAttribute, PTHREAD_PROCESS_SHARED) != 0){
50          printf("Error: set shared attribute mutex\n");
51          return -1;
52      }
53      *Size = 0;
54      if(pthread_mutex_init(Lock, &MutexAttribute) != 0){
55          printf("Error: mutex init\n");
56          return -1;
57      }
58
59      char *filename = NULL;
60      size_t sizename = 0;
61      getline(&filename,&sizename,stdin);
62      filename[strlen(filename)-1] = '\0';
63
64      int id = fork();
65
66      if(id == -1){
67          printf("Error: fork\n");
68          return -1;
69      } else if(id == 0) {
70
71          execl("./child","child",filename,SH_NAME,SH_SIZE_NAME,MUTEX_NAME,(char*) NULL);
72      } else {
73
```

```
74          int num;
75          char sym;
76          if(pthread_mutex_lock(Lock) != 0){
77              printf("Error: mutex lock\n");
78              return -1;
79          }
80          while(scanf("%d%c",&num,&sym) > 0){
81              Data[*Size] = num;
82              *Size += 1;
83              if(sym == '\n'){
84                  if(pthread_mutex_unlock(Lock) != 0){
85                      printf("Error: mutex unlock\n");
86                      return -1;
87                  }
88                  wait(Size,0);
89                  if(pthread_mutex_lock(Lock) != 0){
90                      printf("Error: mutex lock\n");
91                      return -1;
92                  }
93              }
94          }
95          *Size = -1;
96          if(pthread_mutex_unlock(Lock) != 0){
97              printf("Error: mutex unlock\n");
98              return -1;
99          }
100     }
101
102     if(munmap(Data,getpagesize()) != 0){
103         printf("Error: unmap file\n");
104         return -1;
105     }
106     if(munmap(Size,sizeof(int)) != 0){
107         printf("Error: unmap file\n");
108         return -1;
109     }
110     if(munmap(Lock,sizeof(pthread_mutex_t*)) != 0){
111         printf("Error: unmap file\n");
112         return -1;
113     }
114     if(close(fd_shared_data) < 0){
115         printf("Error: close file\n");
116         return -1;
117     }
118     if(close(fd_shared_data_size) < 0){
119         printf("Error: close file\n");
120         return -1;
121     }
122     if(close(fd_mutex) < 0){
```

```
123         printf("Error: close file\n");
124         return -1;
125     }
126 
127     return 0;
128 }
```

child.c

```
1  #include "stdio.h"
2  #include <unistd.h>
3  #include <sys/mman.h>
4  #include <sys/types.h>
5  #include <fcntl.h>
6  #include <pthread.h>
7  
8  
9  int main(int argc,char **argv){
10     if(argc < 5){
11         printf("Arguments error");
12         return 1;
13     }
14     char *filename = argv[1];
15     char *sh_data_name = argv[2];
16     char *sh_data_size_name = argv[3];
17     char *mutex_name = argv[4];
18 
19     int fd_shared_data = shm_open(sh_data_name, O_RDWR | O_CREAT, S_IRWXU);
20     int fd_shared_data_size = shm_open(sh_data_size_name, O_RDWR | O_CREAT, S_IRWXU);
21     int fd_mutex = shm_open(mutex_name,O_RDWR | O_CREAT, S_IRWXU);
22     if(fd_shared_data == -1 || fd_shared_data_size == -1 || fd_mutex == -1){
23         printf("Error: shared memory open\n");
24         return -1;
25     }
26     int *Data = (int*) mmap(NULL,getpagesize(),PROT_READ | PROT_WRITE, MAP_SHARED,
           fd_shared_data, 0);
27     int *Size = (int*) mmap(NULL,sizeof(int),PROT_READ | PROT_WRITE, MAP_SHARED,
           fd_shared_data_size, 0);
28     pthread_mutex_t *Lock = (pthread_mutex_t*) mmap(NULL,sizeof(pthread_mutex_t*),
           PROT_READ | PROT_WRITE, MAP_SHARED,fd_mutex,0);
29     if (Data == MAP_FAILED || Size == MAP_FAILED || Lock == MAP_FAILED) {
30         printf("Error: map file\n");
31         return -1;
32     }
33     FILE *file;
34     file = fopen(filename,"w");
35     if(file == NULL){
36         printf("Error: fopen file\n");
37         return -1;
38     }
```

```
39      while ((*Size) != -1){
40          if(pthread_mutex_lock(Lock) != 0){
41              printf("Error: mutex lock\n");
42              return -1;
43          }
44          if(*Size > 0){
45              long long sum = 0;
46              for(int i = 0;i < *Size; ++i){
47                  sum += Data[i];
48              }
49              *Size = 0;
50              fprintf(file,"%lld\n",sum);
51          }
52          if(pthread_mutex_unlock(Lock) != 0){
53              printf("Error: mutex unlock\n");
54              return -1;
55          }
56      }
57
58      if(fclose(file) != 0){
59          printf("Error: fclose file\n");
60          return -1;
61      }
62      if(munmap(Data,getpagesize()) != 0){
63          printf("Error: unmap file\n");
64          return -1;
65      }
66      if(munmap(Size,sizeof(int)) != 0){
67          printf("Error: unmap file\n");
68          return -1;
69      }
70      if(munmap(Lock,sizeof(pthread_mutex_t*)) != 0){
71          printf("Error: unmap file\n");
72          return -1;
73      }
74      if(close(fd_shared_data) < 0){
75          printf("Error: close file\n");
76          return -1;
77      }
78      if(close(fd_shared_data_size) < 0){
79          printf("Error: close file\n");
80          return -1;
81      }
82      if(close(fd_mutex) < 0){
83          printf("Error: close file\n");
84          return -1;
85      }
86      if(shm_unlink(sh_data_name) != 0){
87          printf("Error: shared memory unlink\n");
```

```
88          return -1;
89      }
90      if(shm_unlink(sh_data_size_name) != 0){
91          printf("Error: shared memory unlink\n");
92          return -1;
93      }
94      if(shm_unlink(mutex_name)!= 0){
95          printf("Error: shared memory unlink\n");
96          return -1;
97      }
98      return 0;
99  }
```

5 Демонстрация работы программы

```
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB4$ make
gcc -c -Wall parent.c
gcc parent.o -pthread -lrt -o parent
gcc -c -Wall child.c
gcc child.o -pthread -lrt -o child
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB4$ ./parent
test
1 2 3 4 5
0 0 0
12 45 34 54
42 -5
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB4$ cat test
15
0
145
37
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB4$ strace -f -e trace=
"%process,read,write,dup2,mmap" -o log.txt ./parent
test2
1 2 3
0 0
2 -1
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB4$ cat log.txt
679   execve("./parent",["./parent"],0x7fffe2329438 /* 29 vars */) = 0
679   arch_prctl(0x3001 /* ARCH_??? */,0x7ffe9622c3c0) = -1
EINVAL (Invalid argument)
679   mmap(NULL,45372,PROT_READ,MAP_PRIVATE,3,0) = 0x7f8283ee3000
```

```
679   read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 7\0
\0\0\0\0\0"...,832) = 832
679   mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE
|MAP_ANONYMOUS,-1,0) = 0x7f8283ee1000
679   mmap(NULL,44000,PROT_READ,MAP_PRIVATE|
MAP_DENYWRITE,3,0) = 0x7f8283ed6000
679   mmap(0x7f8283ed9000,16384,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED
|MAP_DENYWRITE,3,0x3000) = 0x7f8283ed9000
679   mmap(0x7f8283edd000,4096,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x7000) = 0x7f8283edd000
679   mmap(0x7f8283edf000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x8000) = 0x7f8283edf000
679   read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220
\201\0\0\0\0\0\0"...,832) = 832
679   mmap(NULL,140408,PROT_READ,MAP_PRIVATE|
MAP_DENYWRITE,3,0) = 0x7f8283eb3000
679   mmap(0x7f8283eba000,69632,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x7000) = 0x7f8283eba000
679   mmap(0x7f8283ecb000,20480,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x18000) = 0x7f8283ecb000
679   mmap(0x7f8283ed0000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1c000) = 0x7f8283ed0000
679   mmap(0x7f8283ed2000,13432,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7f8283ed2000
679   read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q
\2\0\0\0\0\0"...,832) = 832
679   mmap(NULL,2036952,PROT_READ,MAP_PRIVATE|
MAP_DENYWRITE,3,0) = 0x7f8283cc1000
679   mmap(0x7f8283ce6000,1540096,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x25000) = 0x7f8283ce6000
679   mmap(0x7f8283e5e000,303104,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x19d000) = 0x7f8283e5e000
679   mmap(0x7f8283ea9000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1e7000) = 0x7f8283ea9000
679   mmap(0x7f8283eaf000,13528,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7f8283eaf000
679   mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_ANONYMOUS,-1,0) = 0x7f8283cbe000
679   arch_prctl(ARCH_SET_FS,0x7f8283cbe740) = 0
679   mmap(NULL,4096,PROT_READ|PROT_WRITE,
MAP_SHARED,3,0) = 0x7f8283f1b000
```

```
679   mmap(NULL,4,PROT_READ|PROT_WRITE,MAP_SHARED,4,0) = 0x7f8283eee000
679   mmap(NULL,8,PROT_READ|PROT_WRITE,MAP_SHARED,5,0) = 0x7f8283eed000
679   read(0,"test2\n",1024)          = 6
679   clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|
CLONE_CHILD_SETTID|SIGCHLD,child_tidptr=0x7f8283cbea10) = 680
679   read(0, <unfinished ...>
680   execve("./child",["child","test2","my_shared_mem",
"my_shared_mem_size","my_mutex"],0x7ffe9622c4a8 /* 29 vars */) = 0
680   arch_prctl(0x3001 /* ARCH_??? */,0x7ffc20777f70) = -1
EINVAL (Invalid argument)
680   mmap(NULL,45372,PROT_READ,MAP_PRIVATE,3,0) = 0x7fa203107000
680   read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 7\0
\0\0\0\0\0"...,832) = 832
680   mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_ANONYMOUS,-1,0) = 0x7fa203105000
680   mmap(NULL,44000,PROT_READ,MAP_PRIVATE|
MAP_DENYWRITE,3,0) = 0x7fa2030fa000
680   mmap(0x7fa2030fd000,16384,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x3000) = 0x7fa2030fd000
680   mmap(0x7fa203101000,4096,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x7000) = 0x7fa203101000
680   mmap(0x7fa203103000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x8000) = 0x7fa203103000
680   read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220
\201\0\0\0\0\0\0"...,832) = 832
680   mmap(NULL,140408,PROT_READ,MAP_PRIVATE|
MAP_DENYWRITE,3,0) = 0x7fa2030d7000
680   mmap(0x7fa2030de000,69632,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x7000) = 0x7fa2030de000
680   mmap(0x7fa2030ef000,20480,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x18000) = 0x7fa2030ef000
680   mmap(0x7fa2030f4000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1c000) = 0x7fa2030f4000
680   mmap(0x7fa2030f6000,13432,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7fa2030f6000
680   read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q
\2\0\0\0\0\0"...,832) = 832
680   mmap(NULL,2036952,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0)
= 0x7fa202ee5000
680   mmap(0x7fa202f0a000,1540096,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x25000) = 0x7fa202f0a000
```

```
680   mmap(0x7fa203082000,303104,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x19d000) = 0x7fa203082000
680   mmap(0x7fa2030cd000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1e7000) = 0x7fa2030cd000
680   mmap(0x7fa2030d3000,13528,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7fa2030d3000
680   mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_ANONYMOUS,-1,0) = 0x7fa202ee2000
680   arch_prctl(ARCH_SET_FS,0x7fa202ee2740) = 0
680   mmap(NULL,4096,PROT_READ|PROT_WRITE,
MAP_SHARED,3,0) = 0x7fa20313f000
680   mmap(NULL,4,PROT_READ|PROT_WRITE,MAP_SHARED,4,0) = 0x7fa203112000
680   mmap(NULL,8,PROT_READ|PROT_WRITE,MAP_SHARED,5,0) = 0x7fa203111000
679   <... read resumed>"1 2 3\n",1024) = 6
679   read(0,"0 0\n",1024)            = 4
679   read(0,"2 -1\n",1024)           = 5
679   read(0,"",1024)                 = 0
680   write(6,"6\n0\n1\n",6)          = 6
679   exit_group(0)                     = ?
679   +++ exited with 0 +++
680   exit_group(0)                     = ?
680   +++ exited with 0 +++
```

6 Вывод

Взаимодействие между процессами можно организовать при помощи каналов, сокетов и отображаемых файлов. В данной лабораторной работе был изучен и применен механизм межпроцессорного взаимодействия – file mapping. Файл отображается на оперативную память таким образом, что мы можем взаимодействовать с ним как с массивом.

Благодаря этому вместо медленных запросов на чтение и запись мы выполняем отображение файла в ОЗУ и получаем произвольный доступ за O(1). Из-за этого при использовании этой технологии межпроцессорного взаимодействия мы можем получить ускорении работы программы, в сравнении, с использованием каналов.

Из недостатков данного метода можно выделить то, что дочерние процессы обязательно должны знать имя отображаемого файла и также самостоятельно выполнить отображение.