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: С. А. Арапов
Преподаватель: Е. С Миронов
Группа: М8О-208Б-19
Дата: Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Тема: Приобритение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задача: 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.

Группа вариантов №1: Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Вариант №4: Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float.

1 Описание программы

Код программы написан на языке Си, в ОС на базе ядра Linux. Во время выполнения программы создаётся дочерний процесс, данные в который передаются с помощью shared memory. Дочерний процесс берёт первое число в строке и делит на последующие, после чего записывает результат в файл. Имя файла задаётся в первой строке при запуске программы.

2 Алгоритм решения

В родительском процессе создаётся shared memory объекты, получаем дескрипторы этих объектов. Объекты отвечают за данные, размер этих данных и мьютекс. После чего файлы проецируются в память и инициализируются. Далее пользуемся функцией fork() для создания дочернего процесса. В дочерний процесс передаём имя файла,куда записываются вычисления, и наши shared memory объекты.
Родитель блокирует мьютекс, считывает число, символ после числа и увеличивает размер данных на единицу. Если символом оказался перенос строки, то мьютекс разблокируется до тех пор пока размер данных снова не станет равен нулю. Если число оказалось нулём то размер данных становится равным минус единице и разблокируется мьютекс. Дочерний процесс всё это время пытается заблокировать мьютес, если это все же произоёдет до ввода родителя, то он разблокируется и процесс повторится. После того как всё же совершается ввод то производим поочерёдное деление первого числа из наших данных на все остальные. Если длина данных оказалась -1, то есть происходит деление на 0 то выходим из цикла. В конце закрываем файлы и отключаем проекции в двух программах.

3 Исходный код

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

```
119         printf("Error: close file\n");
120         return -1;
121     }
122     if(close(fd_shared_data_size) < 0){
123         printf("Error: close file\n");
124         return -1;
125     }
126     if(close(fd_mutex) < 0){
127         printf("Error: close file\n");
128         return -1;
129     }
130 
131     return 0;
132 }
```

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     float *Data = (float*) 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
40      while ((*Size) != -1){
41          if(pthread_mutex_lock(Lock) != 0){
42              printf("Error: mutex lock\n");
43              return -1;
44          }
45
46          if(*Size > 0){
47              float result = Data[0];
48              for(int i = 1;i < *Size; ++i){
49                  if(*Size == -1){
50                      break;
51                  }
52                  result /= Data[i];
53              }
54              *Size = 0;
55              fprintf(file,"%f\n",result);
56          }
57          if(pthread_mutex_unlock(Lock) != 0){
58              printf("Error: mutex unlock\n");
59              return -1;
60          }
61      }
62
63      if(fclose(file) != 0){
64          printf("Error: fclose file\n");
65          return -1;
66      }
67      if(munmap(Data,getpagesize()) != 0){
68          printf("Error: unmap file\n");
69          return -1;
70      }
71      if(munmap(Size,sizeof(int)) != 0){
72          printf("Error: unmap file\n");
73          return -1;
74      }
75      if(munmap(Lock,sizeof(pthread_mutex_t*)) != 0){
76          printf("Error: unmap file\n");
77          return -1;
78      }
79      if(close(fd_shared_data) < 0){
```

```
80          printf("Error: close file\n");
81          return -1;
82      }
83      if(close(fd_shared_data_size) < 0){
84          printf("Error: close file\n");
85          return -1;
86      }
87      if(close(fd_mutex) < 0){
88          printf("Error: close file\n");
89          return -1;
90      }
91      if(shm_unlink(sh_data_name) != 0){
92          printf("Error: shared memory unlink\n");
93          return -1;
94      }
95      if(shm_unlink(sh_data_size_name) != 0){
96          printf("Error: shared memory unlink\n");
97          return -1;
98      }
99      if(shm_unlink(mutex_name)!= 0){
100         printf("Error: shared memory unlink\n");
101         return -1;
102     }
103     return 0;
104 }
```

makefile

```
1  CC = gcc
2  CFLAGS = -c -Wall
3  FINALFLAGS = -pthread -lrt -o
4
5  all: parent child
6
7  parent: parent.o
8    $(CC) parent.o $(FINALFLAGS) parent
9  parent.o: parent.c
10   $(CC) $(CFLAGS) parent.c
11 child: child.o
12   $(CC) child.o $(FINALFLAGS) child
13 child.o: child.c
14   $(CC) $(CFLAGS) child.c
15 clean:
16   rm -r *.o parent child
```

4 Запуск программы и демонстрация работы

```
ogrocket@LAPTOP-ADULJM7A:/mnt/d/OS/OS/os_lab4$ ls
```

```
child.c  makefile  parent.c
ogrocket@LAPTOP-ADULJM7A:/mnt/d/OS/OS/os_lab4$ 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
ogrocket@LAPTOP-ADULJM7A:/mnt/d/OS/OS/os_lab4$ ./parent
test
8421 8281 48128 4812 8
4218481 1
11 1 1 1
1
1 10
1429 1
192 124 0
ogrocket@LAPTOP-ADULJM7A:/mnt/d/OS/OS/os_lab4$ cat test
0.000000
4218481.000000
11.000000
0.100000
1429.000000

ogrocket@LAPTOP-ADULJM7A:/mnt/d/OS/OS/os_lab4$
strace -f -e trace="%process,read,write,dup2,mmap" -o log ./parent
test
4291 82184.124
42149 111
1111 1 11 111.1111
124124 1421
4129421 139.213213 0.5 125
ogrocket@LAPTOP-ADULJM7A:/mnt/d/OS/OS/os_lab4$ cat log
63    execve("./parent",["./parent"],0x7fffdfe14408 /* 19 vars */) = 0
63    arch_prctl(0x3001 /* ARCH_??? */,0x7fffd7d6cfb0) = -1 EINVAL (Invalid
argument)
63    mmap(NULL,33800,PROT_READ,MAP_PRIVATE,3,0) = 0x7fcc52843000
63    read(3,"77ELF>7"...,832) = 832
63    mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) =
0x7fcc52840000
63    mmap(NULL,44000,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7fcc52800000
63    mmap(0x7fcc52803000,16384,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x3000) = 0x7fcc52803000
```

```
63    mmap(0x7fcc52807000,4096,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x7000) = 0x7fcc52807000
63    mmap(0x7fcc52809000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x8000) = 0x7fcc52809000
63    read(3,"77ELF>2001"...,832) = 832
63    mmap(NULL,140408,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7fcc527dd000
63    mmap(0x7fcc527e4000,69632,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x7000) = 0x7fcc527e4000
63    mmap(0x7fcc527f5000,20480,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x18000) = 0x7fcc527f5000
63    mmap(0x7fcc527fa000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1c000) = 0x7fcc527fa000
63    mmap(0x7fcc527fc000,13432,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7fcc527fc000
63    read(3,"77ELF>60q"...,832) = 832
63    mmap(NULL,2036952,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7fcc525e0000
63    mmap(0x7fcc52605000,1540096,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x25000) = 0x7fcc52605000
63    mmap(0x7fcc5277d000,303104,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x19d000) = 0x7fcc5277d000
63    mmap(0x7fcc527c8000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1e7000) = 0x7fcc527c8000
63    mmap(0x7fcc527ce000,13528,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7fcc527ce000
63    mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_ANONYMOUS,-1,0) = 0x7fcc525d0000
63    arch_prctl(ARCH_SET_FS,0x7fcc525d0740) = 0
63    mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_SHARED,3,0) = 0x7fcc5284b000
63    mmap(NULL,4,PROT_READ|PROT_WRITE,MAP_SHARED,4,0) = 0x7fcc5284a000
63    mmap(NULL,8,PROT_READ|PROT_WRITE,MAP_SHARED,5,0) = 0x7fcc52849000
63    read(0,"test",4096)           = 5
63    clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD,child_tidptr=0x7fcc525d0a10) = 64
63    read(0, <unfinished ...>
64    execve("./child",["child","test","my_shared_mem",
"my_shared_mem_size","my_mutex"],0x7fffd7d6d098 /* 19 vars */) = 0
64    arch_prctl(0x3001 /* ARCH_??? */,0x7ffffa680350) = -1 EINVAL (Invalid
argument)
64    mmap(NULL,33800,PROT_READ,MAP_PRIVATE,3,0) = 0x7f494cda7000
64    read(3,"77ELF>7"...,832) = 832
64    mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
```

```
MAP_ANONYMOUS,-1,0) = 0x7f494cde0000
64    mmap(NULL,44000,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7f494cd90000
64    mmap(0x7f494cd93000,16384,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x3000) = 0x7f494cd93000
64    mmap(0x7f494cd97000,4096,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x7000) = 0x7f494cd97000
64    mmap(0x7f494cd99000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x8000) = 0x7f494cd99000
64    read(3,"77ELF>2001"...,832) = 832
64    mmap(NULL,140408,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7f494cd6d000
64    mmap(0x7f494cd74000,69632,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x7000) = 0x7f494cd74000
64    mmap(0x7f494cd85000,20480,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x18000) = 0x7f494cd85000
64    mmap(0x7f494cd8a000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1c000) = 0x7f494cd8a000
64    mmap(0x7f494cd8c000,13432,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7f494cd8c000
64    read(3,"77ELF>60q"...,832) = 832
64    mmap(NULL,2036952,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7f494cb70000
64    mmap(0x7f494cb95000,1540096,PROT_READ|PROT_EXEC,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x25000) = 0x7f494cb95000
64    mmap(0x7f494cd0d000,303104,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x19d000) = 0x7f494cd0d000
64    mmap(0x7f494cd58000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,3,0x1e7000) = 0x7f494cd58000
64    mmap(0x7f494cd5e000,13528,PROT_READ|PROT_WRITE,MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS,-1,0) = 0x7f494cd5e000
64    mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
= 0x7f494cda0000
64    arch_prctl(ARCH_SET_FS,0x7f494cda0740) = 0
64    mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_SHARED,3,0) = 0x7f494cddc000
64    mmap(NULL,4,PROT_READ|PROT_WRITE,MAP_SHARED,4,0) = 0x7f494cdaf000
64    mmap(NULL,8,PROT_READ|PROT_WRITE,MAP_SHARED,5,0) = 0x7f494cdae000
63    <... read resumed>"4291 82184.124",4096) = 15
63    read(0,"42149 111",4096)      = 10
63    read(0,"1111 1 11 111.1111",4096) = 19
63    read(0,"124124 1421",4096)    = 12
63    read(0,"4129421 139.213213 0.5 125",4096) = 27
63    read(0,"",4096)                 = 0
64    write(6,"0.052212379.7207340.90900087."...,50) = 50
```

```
63    exit_group(0)                     = ?
63    +++ exited with 0 +++
64    exit_group(0)                     = ?
64    +++ exited with 0 +++
```

5 Выводы

Выполнив данную лабораторную работу я познакомился с механизмом межпроцессорного взаимодействия, который называется «file mapping». Особенность этой технологии в том, что файл отображается на оперативную память, причем таким образом, чтобы была возможность взаимодействия с ним как с обычным массивом. Такой подход позволяет получить доступ за $O(1)$, это существенный плюс данного подхода.