Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Операционные системы»

Студент: А. П. Шорохов
Преподаватель: Е. С. Миронов
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2020

1 Описание

Необходимо разработать 3 программы. Назовём их **A, B и C**. **A** принимает из стандартного потока ввода строки, а далее отправляет их программе **C**. Отправка строк должна производиться построчно. Программа **C** печатает в стандартный вывод полученнкую строку от программы **A**. После получения программа **C** отправляет программе **A** сообщение о том, что строка получена. До тех пор, пока программа **A** не примет «сообщение о получении строки» от программы **C**, она не может отправлять следующую строку программе **C**.
Программа **B** пишет в стандартный поток вывода количество отправленных символов программой **A** и количество принятых символов программой **C**. Данную информацию программа **B** получает от программ **A** и **C** соответственно.

2 Алгоритм

Для решения поставленной задачи мы будем использовать разделяемую память, отображение файла в память и семафоры.

Распишем алгоритм по пунктам:
1. **A** создаёт необходимые семафоры, объект разделяемой памяти, отображает 100 байт в память. Считывает строку со стандартного потока, затем выполняет системный вызов и создает дочерний процесс.

2. Дочерний процесс создаёт ещё один процесс, который запускает программу **B**. Программа **B** в свою очередь подсчитывает длину строки, которую передала программа **A**

3. После выполнения процесса **B** выполняется процесс **C**, в котором так же присутствует вызов для создания дочернего процесса **B**.

4. **B** подсчитывает длину строки, которую получил **C**.

5. Процесс **A** дожидается завершения дочерних процессов, после чего считывает новую строку со стандартного потока.

3 Исходный код

Программа A.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/mman.h>

#define BUF_SIZE 100
#define SHARED_MEMORY_NAME "/shm_file"
#define FIRST_SEM "/sem1"
#define SECOND_SEM "/sem2"
#define THIRD_SEM "/sem3"

int main() {




  int fd_shm;
  char* shmem;
  char* tmp = (char*)malloc(sizeof(char) * BUF_SIZE);
  char* buf_size = (char*)malloc(sizeof(char) * 10);


  sem_t* sem1 = sem_open(FIRST_SEM, O_CREAT, 0660, 0);
  sem_t* sem2 = sem_open(SECOND_SEM, O_CREAT, 0660, 0);
  sem_t* sem3 = sem_open(THIRD_SEM, O_CREAT, 0660, 0);
  if (sem1 == SEM_FAILED || sem2 == SEM_FAILED || sem3 == SEM_FAILED) {
    perror("Semaphore opening error, program 'a'\n");
    exit(1);
  }


  if (shm_unlink(SHARED_MEMORY_NAME) == -1) {
    perror("shm_unlink error\n");
    exit(1);
  }

  // Get shared memory obj
  if ((fd_shm = shm_open(SHARED_MEMORY_NAME, O_RDWR | O_CREAT | O_EXCL, 0660)) == -1)
      {
    perror("shm_open error, program 'a'\n");
```

```
46      exit(1);
47    }
48    // Allocate memory for shm obj
49    if (ftruncate(fd_shm, BUF_SIZE) == -1) {
50      perror("ftruncate error, program 'a'\n");
51      exit(-1);
52    }
53
54
55    // Create a new mapping
56    shmem = (char*)mmap(NULL, BUF_SIZE, PROT_WRITE | PROT_READ, MAP_SHARED, fd_shm, 0);
57
58
59    //convert file descriptor to string
60    sprintf(buf_size, "%d", BUF_SIZE);
61
62
63
64    char* argv[] = { buf_size, SHARED_MEMORY_NAME, SECOND_SEM, THIRD_SEM, NULL };
65
66
67
68
69    while (scanf ("%s", tmp)) {
70
71    pid_t p = fork();
72    if (p == 0) {
73      pid_t p_1 = fork();
74      if (p_1 == 0) {
75        sem_wait(sem1);
76        printf("program a sent:\n");
77        if (execve("./b.out", argv, NULL) == -1) {
78          perror("Could not execve, program 'a'\n");
79        }
80      } else if (p_1 > 0) {
81        sem_wait(sem3);
82        if (execve("./c.out", argv, NULL) == -1) {
83          perror("Could not execve, program 'a'\n");
84        }
85      }
86
87
88    } else if (p > 0) {
89
90      sprintf(shmem, "%s", tmp);
91      sem_post(sem1);
92      sem_wait(sem2);
93      printf("###############\n\n");
94
```

```
  }

}


shm_unlink(SHARED_MEMORY_NAME);
sem_unlink(FIRST_SEM);
sem_unlink(SECOND_SEM);
sem_unlink(THIRD_SEM);
sem_close(sem1);
sem_close(sem2);
sem_close(sem3);
close(fd_shm);


}
```

Программа B.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/mman.h>

int main(int argc, char const * argv[]) {
  if (argc < 2) {
    perror("not too much arg, program 'b'\n");
    exit(1);
  }
  int buf_size = atoi(argv[0]);
  char const* shared_memory_name = argv[1];
  char const* sem3_name = argv[3];
  int fd_shm;

  if ((fd_shm = shm_open(shared_memory_name, O_RDWR , 0660)) == -1) {
    perror("shm_open error, program 'b'\n");
    exit(1);
  }

  sem_t* sem3 = sem_open(sem3_name, 0,0,0);
  if (sem3 == SEM_FAILED) {
    perror("sem3 error, program 'b'\n");
    exit(1);
  }
```

```
32
33    char* shmem = (char*)mmap(NULL, buf_size, PROT_WRITE | PROT_READ, MAP_SHARED, fd_shm
          , 0);
34    int size = strlen(shmem);
35    printf("%d symbols\n", size);
36    sem_post(sem3);
37  }
```

Программа C.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <sys/types.h>
4   #include <sys/stat.h>
5   #include <sys/wait.h>
6   #include <fcntl.h>
7   #include <string.h>
8   #include <unistd.h>
9   #include <semaphore.h>
10  #include <sys/mman.h>
11
12
13  int main(int argc, char* const argv[])
14  {
15
16    if (argc < 2) {
17      printf("not to much arg, program 'c'\n");
18      return 0;
19    }
20
21    int buf_size = atoi(argv[0]);
22    char const* shared_memory_name = argv[1];
23    char const* sem2_name = argv[2];
24    char const* sem3_name = argv[3];
25    int fd_shm;
26
27
28
29
30    if ((fd_shm = shm_open(shared_memory_name, O_RDWR , 0660)) == -1) {
31      perror("shm_open error, program 'c'\n");
32      exit(1);
33    }
34
35    sem_t* sem2 = sem_open(sem2_name, 0,0,0);
36    sem_t* sem3 = sem_open(sem3_name, 0,0,0);
37    if (sem2 == SEM_FAILED || sem3 == SEM_FAILED) {
38      perror("sem2 error, program 'c'\n");
39      exit(1);
40    }
```

```
41
42    char* shmem = (char*)mmap(NULL, buf_size, PROT_WRITE | PROT_READ, MAP_SHARED, fd_shm
          , 0);
43    pid_t p = fork();
44    if (p == 0) {
45      printf("program c take:\n");
46      if (execve("b.out", argv, NULL) == -1) {
47        perror("execve error, program 'c'\n");
48        exit(1);
49      }
50    } else if (p > 0) {
51      sem_wait(sem3);
52      printf("%s\n", shmem);
53    }
54
55    sem_post(sem2);
56  }
```

Файл makefile

```
1  KEYS=-lrt -lpthread
2
3  all: a.c c.c
4  gcc a.c -o a.out $(KEYS)
5  gcc c.c -o c.out $(KEYS)
6  gcc b.c -o b.out $(KEYS)
7
8  a: a.c
9  gcc a.c -o a.out $(KEYS)
10
11 b: b.c
12 gcc b.c -o b.out $(KEYS)
13
14 c: c.c
15 gcc c.c -o c.out $(KEYS)
```

4 Консоль

```
dobriy_alien@LAPTOP-2MM4OK81:/mnt/c/Users/shoro/Desktop/MAI/2course/os/cp3/os_kp$
make
gcc a.c -o a.out -lrt -lpthread
gcc c.c -o c.out -lrt -lpthread
gcc b.c -o b.out -lrt -lpthread
dobriy_alien@LAPTOP-2MM4OK81:/mnt/c/Users/shoro/Desktop/MAI/2course/os/cp3/os_kp$
./a.out
Bitcoin
program A sent:
7 symbols
program C got:
7 symbols
Bitcoin
-----------------

Ethereum
program A sent:
8 symbols
program C got:
8 symbols
Ethereum
-----------------

Proof-of-Stake
program A sent:
14 symbols
program C got:
14 symbols
Proof-of-Stake
-----------------

Satoshi Nakamoto
program A sent:
7 symbols
program C got:
7 symbols
Satoshi
-----------------
```

```
program A sent:
8 symbols
program C got:
8 symbols
Nakamoto
-----------------
```

5 Выводы

В процессе выполнения курсового проекта я написал многопроцессорную программу, применив знания, полученные в ходе изучения курса «Операционные системы». Задача показалась мне довольно интересной и ее реализация была не очень сложной. Использованные в данной программе отображение файлов в память и разделяемая память очень удобны и могут быть применимы и в других многопоточных программах.