CSE344 homework 3

Ümit Altıntaş
171044005

April 25, 2021

1 USAGE

I got the command line arguments with getopt(), also I have moved the usage print function to the usage.h header for further changing.

2 VARIABLES

I have defined main variables as global for easy memory management. Moved them to the global header file.

```
#define PID_INDEX 0
#define POTATO_INDEX 1
#define MAX_FIFO_C 250
#define FINISH_SIGN (-3)

typedef struct SM {
  int potato_c;
  sem_t sem;   umitaltintas, 23.04.2021 02:
  short fifo_index;
  pid_t peer[MAX_FIFO_C][2];
} sm;
sem_t *sem;
char fifo_file_names[MAX_FIFO_C][250];
int fifo_fd[MAX_FIFO_C];
int read_index;
int fifo_count = 0;
caddr_t memory_pointer;
short is_creator = 1;
sm *shared_memory_pointer;
char *shared_memory_name;
char *fifo_names_file;
int pot_sw_count;
char *sem_name;

#endif // HW3_GLOBALS_H
```

Figure 1: globals figure

3 SIGNAL

I have define a finish function as a signal handler.

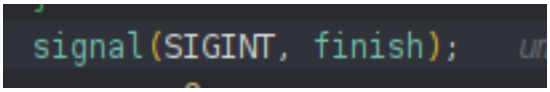

Figure 2: signal figure

4 SHARED MEMORY

First of all I have tried to open file with EXCL flag so if it is already exist return -1, if it is return -1 I have tried without EXCL flag . With this return value have decided who will truncate it and create default values of shared memory. Lastly all processes write their information to the shared memory.(potato sw number eg.)

```
int fd = shm_open(shared_memory_name,  oflag: O_RDWR | O_CREAT | O_EXCL,
                  mode: S_IRUSR | S_IWUSR);

if (fd == -1) {
  if (errno == EEXIST) {
    is_creator = 0;
    if (1 > (fd = shm_open(shared_memory_name, O_RDWR,  mode: S_IRUSR | S_IWUSR))) {
      report_and_exit( msg: "can't open shared memory");
    }
  } else {
    report_and_exit( msg: "can't open shared memory");
  }
} else {   umitaltintas, Yesterday • finish project
  if ((ftruncate(fd, sizeof(struct SM)) == -1)) {
    report_and_exit( msg: "ftruncate failed");
  }
}

memory_pointer = mmap( addr: NULL, /* let system pick where to put segment */
                       sizeof(struct SM),      /* how many bytes */
                       prot: PROT_READ | PROT_WRITE, /* access protections */
                       MAP_SHARED, /* mapping visible to other processes */
                       fd, /* file descriptor */  offset: 0);
```

Figure 3: shared memory figure

5 SEMAPHORE

I have used to semaphore for synchronization. One for decide who take which fifo file and one for who will write to the shared memory

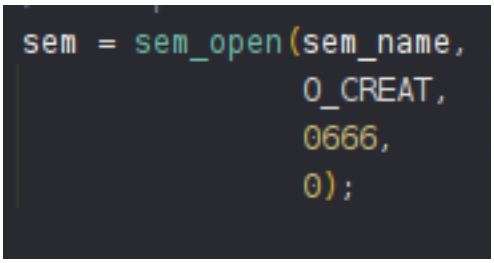

Figure 4: semaphore figure 1

This semaphore used for selecting fifo files. it start with 0. after creator process create fifos it post the semaphore. After that they take their fifos respectively. For desicion who take which fifo i have used a index insede the shared memory.

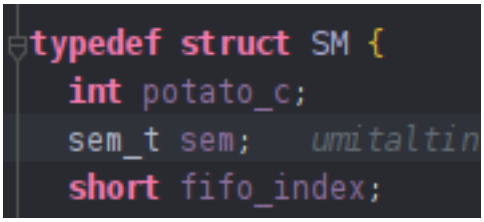

Figure 5: fifo index figure 2

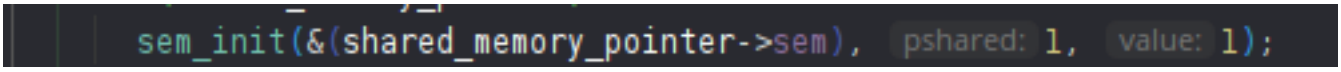

Figure 6: semaphore figure 2

This semaphore decides who can change shared memories values.

6 FIFO

After creation fifos they open their fifos as reader and open others as writer. Also fifo file names index and pid indexes(inside the shared memory) are same.

```
for (int i = 0; i < fifo_count; i++) {
  if (read_index != i) {
    fifo_fd[i] = open(fifo_file_names[i], O_WRONLY);
  } else {
    fifo_fd[i] = open(fifo_file_names[i], O_RDONLY);
  }
}
```

Figure 7: fifo figure 2

7 TRANSFER

At first they wait for semaphore 2 for getting info's from shared memory. after that if they check their potatoes switch number. if it is not zero select a random fifo and send its potato. After sending potato post the semaphore and starts reading its fifo. If they read finish signal they they break the loop and call the finish function for memory management, if it is read a valid potato, it wait for semaphore 2 for changing shared memory. When take permission it decrease the potato switch number and check active potato switch number . If switch number become 0 it decrease active potato count after that it check active potato count also if it become 0 sends all fifos a finish signal which is -3 in my case. All of the above is inside a finite loop.

```
while (true) {

    // write potato to fifo
    if (-1 == sem_wait(&shared_memory_pointer->sem)) {
      report_and_exit("sem_wait");
    }
    if (shared_memory_pointer->peer[potato_id][POTATO_INDEX]) {
      random_number = select_random_index();
      printf("pid=%d sending potato number %d to %s; %d switches left\n",
             getpid(), shared_memory_pointer->peer[potato_id][PID_INDEX],
             fifo_file_names[random_number],
             shared_memory_pointer->peer[potato_id][POTATO_INDEX] - 1);
      fflush(stdout);
      write(fifo_fd[random_number], &potato_id, sizeof(int));
    }
    if (-1 == sem_post(&shared_memory_pointer->sem)) {
      report_and_exit("sem_post");
    }

    // read potato from fifo
    read(fifo_fd[read_index], &potato_id, sizeof(int));
    if (potato_id == FINISH_SIGN) {
      break;
    } else {

      // update shared memory
      if (-1 == sem_wait(&shared_memory_pointer->sem)) {
        report_and_exit("sem_wait");
      }
      printf("pid=%d receiving potato number %d from %s\n", getpid(),
             shared_memory_pointer->peer[potato_id][PID_INDEX],
             fifo_file_names[read_index]);
      // update switch count
      shared_memory_pointer->peer[potato_id][POTATO_INDEX]--;

      // update potato count
      if (shared_memory_pointer->peer[potato_id][POTATO_INDEX] == 0) {
        printf("pid=%d; potato number %d has cooled down.\n", getpid(),
               shared_memory_pointer->peer[potato_id][PID_INDEX]);
        shared_memory_pointer->potato_c--;
      }
      // handle finish case
      if (shared_memory_pointer->potato_c == 0) {
        send_finish_sign();
        if (-1 == sem_post(&shared_memory_pointer->sem)) {
          report_and_exit("sem_post");
        }
        break;
      }
      if (-1 == sem_post(&shared_memory_pointer->sem)) {
        report_and_exit("sem_post");
```

8 MEMORY

Using advantage of the defining main variables as global. I can easyly free and close them with finish function.

```c
void finish(int sig) {
    sem_unlink(sem_name);
    sem_close(sem);
    sem_close(&shared_memory_pointer->sem);
    shm_unlink(shared_memory_name);
    munmap(shared_memory_pointer, sizeof(sm));
    exit(sig);
}
```

Figure 8: memory figure 2

9 FILE STRUCTURE

-src –globals.h –usage.h –usage.c –main.c -makefile -report.pdf -latex –main.lat