Excercise Sheet 4

Johannes Koch

May 3, 2018

1 Task 1

Compile `reader.c` and `task1.c`:
`gcc -Wall -Werror -std=c99 reader.c -o reader`
`gcc -Wall -Werror -std=c99 task1.c -o task1`
and call:
`./reader & ./task1` or each of the binarys in a separate console window.

1.1 `reader.c`

```c
#define _POSIX_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <string.h>
#include <fcntl.h>

#define BUF_SIZE 16
#define SHM_SIZE 16
#define FIFO "RESULT_FIFO"
#define SHM_KEY 666

int main(void) {
  // needs to be int* since shmat() won't accept &int
  int shm_id, *counter;
  key_t key = SHM_KEY;

  fprintf(stdout, "%d: creating shared memory segment... ", getpid());

  if( (shm_id = shmget(key, SHM_SIZE, IPC_CREAT | 0666)) < 0 ){
    perror("shmget");
    return EXIT_FAILURE;
  }
```

```
if( (counter = shmat(shm_id, NULL, 0)) < 0 ){
  perror("shmat");
  return EXIT_FAILURE;
}

*counter = 0;

fprintf(stdout, "done\n%d: creating fifo... ", getpid());

FILE* fifo_fp;

unlink(FIFO);

if( mkfifo(FIFO, 0777) == -1){
  perror("mkfifo");
  return EXIT_FAILURE;
}

fprintf(stdout, "done\n%d: waiting for fifo...\n", getpid());

if( (fifo_fp = fopen(FIFO, "r")) == NULL ){
  perror("fopen");
  return EXIT_FAILURE;
}

// read from fifo to buffer, convert buffer to long
char buf[BUF_SIZE], *end;
if( read(fileno(fifo_fp), buf, sizeof(buf)) > 0){
  fprintf(stdout, "%d: received something: %ld\n", getpid(), strtol(buf
  , &end, 10));
}

// close fifo
if( fclose(fifo_fp) < 0){
  perror("fclose");
  return EXIT_FAILURE;
}

// unlink fifo
if( unlink(FIFO) == -1){
  perror("unlink");
  return EXIT_FAILURE;
}

// detatch counter from shared memory
if( shmdt(counter) < 0){
  perror("shmdt");
  return EXIT_FAILURE;
}

// remove shared memory when all processes detatched
if( shmctl(shm_id, IPC_RMID, 0) < 0){
  perror("shmctl");
```

```
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

1.2 task1.c

```
#define _POSIX_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>

#define SHM_SIZE 16
#define FIFO "RESULT_FIFO"
#define SHM_KEY 666
#define CHILDREN 100
#define MAX 100

int main(void) {
  int shm_id, *counter;
  key_t key = SHM_KEY;

  fprintf(stdout, "%d: accessing shared memory segment... ", getpid());

  if( (shm_id = shmget(key, SHM_SIZE, IPC_CREAT | 0666)) < 0 ){
    perror("shmget");
    _exit(EXIT_FAILURE);
  }

  if( (counter = shmat(shm_id, NULL, 0)) < 0 ){
    perror("shmat");
    _exit(EXIT_FAILURE);
  }

  fprintf(stdout, "done\n%d: spawing %d child processes... ", getpid(),
   CHILDREN);

  pid_t cpid;
  for(int i = 0; i < CHILDREN; i++){
    if( (cpid = fork()) < 0){
      perror("fork");
      _exit(EXIT_FAILURE);
    } else if( cpid == 0 ){
      for(int j = 0; j < MAX; j++){
        ++*counter;
      }
      _exit(EXIT_SUCCESS);
```

```
    }
  }

  fprintf(stdout, "done\n%d: waiting for children... ", getpid());

  while( wait(NULL) > 0);

  fprintf(stdout, "done\n%d: printing result: %d\n", getpid(), *counter);

  fprintf(stdout, "%d: writing to pipe... ", getpid());
  FILE* fifo_fp;

  if( (fifo_fp = fopen(FIFO, "w")) == NULL ){
    perror("fopen");
    _exit(EXIT_FAILURE);
  }

  fprintf(fifo_fp, "%d", *counter);

  if( fclose(fifo_fp) < 0 ){
    perror("fclose");
    return EXIT_FAILURE;
  }

  if( shmdt(counter) < 0){
    perror("shmdt");
    return EXIT_FAILURE;
  }

  if( shmctl(shm_id, IPC_RMID, 0) < 0){
    perror("shmctl");
    return EXIT_FAILURE;
  }

  fprintf(stdout, "done\n");

  return EXIT_SUCCESS;
}
```

2 Task 2

Task 2 uses the reader of task 1, `task2a.c` and `task2b.c` both need the additional `-pthread` compile flag. To execute the program, call `./reader & ./task2a`/`./reader & ./task2b` or each of the binarys in a separate console window.
The semaphores in `task2a` seem to be very unreliable(no idea what was done wrong), thats why `task2b` exists.

2.1 task2a.c

```
#define _POSIX_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <semaphore.h> // needs -pthreads compile flag
#include <fcntl.h>

#define SHM_SIZE 16
#define FIFO "RESULT_FIFO"
#define SHM_KEY 666
#define CHILDREN 100
#define MAX 100

int main(void) {

  fprintf(stdout, "%d: accessing shared memory segment... ", getpid());

  int shm_id, *counter;
  key_t key = SHM_KEY;

  if( (shm_id = shmget(key, SHM_SIZE, IPC_CREAT | 0666)) < 0 ){
    perror("shmget");
    _exit(EXIT_FAILURE);
  }

  if( (counter = shmat(shm_id, NULL, 0)) < 0 ){
    perror("shmat");
    _exit(EXIT_FAILURE);
  }


  fprintf(stdout, "done\n%d: creating semaphore... ", getpid());

  sem_t *sem;

  if( (sem = shmat(shm_id, NULL, 0)) < 0){
```

```
  perror("shmat");
  _exit(EXIT_FAILURE);
}

if( sem_init(sem, 1, 1) < 0){
  perror("sem_init");
  _exit(EXIT_FAILURE);
}

fprintf(stdout, "done\n%d: spawing %d child processes... ", getpid(),
 CHILDREN);

pid_t cpid;
for(int i = 0; i < CHILDREN; i++){
  if( (cpid = fork()) < 0){
    sem_destroy(sem);
    perror("fork");
    _exit(EXIT_FAILURE);
  } else if( cpid == 0 ){
    for(int j = 0; j < MAX; j++){
      sem_wait(sem);
      fprintf(stdout, "\t%d: entering critical region... ", getpid());
      ++*counter;
      fprintf(stdout, "leaving now\n");
      sem_post(sem);
    }
    sem_destroy(sem);
    _exit(EXIT_SUCCESS);
  }
}

fprintf(stdout, "done\n%d: waiting for children... ", getpid());

while( wait(NULL) > 0);

if( sem_destroy(sem) < 0){
  perror("sem_destroy");
  return EXIT_FAILURE;
}

fprintf(stdout, "done\n%d: printing result: %d\n", getpid(), *counter);

fprintf(stdout, "%d: writing to pipe... ", getpid());
FILE* fifo_fp;

if( (fifo_fp = fopen(FIFO, "w")) == NULL ){
  perror("fopen");
  _exit(EXIT_FAILURE);
}

fprintf(fifo_fp, "%d", *counter);

if( fclose(fifo_fp) < 0 ){
```

```
    perror("fclose");
    return EXIT_FAILURE;
  }

  fprintf(stdout, "done\n");

  if( shmdt(counter) < 0){
    perror("shmdt");
    return EXIT_FAILURE;
  }

  if( shmdt(sem) < 0){
    perror("shmdt");
    return EXIT_FAILURE;
  }

  if( shmctl(shm_id, IPC_RMID, 0) < 0){
    perror("shmctl");
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

2.2 task2b.c

```
#define _POSIX_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <semaphore.h> // needs -pthread compile flag
#include <fcntl.h>

#define SHM_SIZE 16
#define FIFO "RESULT_FIFO"
#define SHM_KEY 666
#define CHILDREN 100
#define MAX 100
#define SEM "sem"

int main(void) {

  fprintf(stdout, "%d: accessing shared memory segment... ", getpid());

  int shm_id, *counter;
  key_t key = SHM_KEY;

  if( (shm_id = shmget(key, SHM_SIZE, IPC_CREAT | 0666)) < 0 ){
    perror("shmget");
```

```
  _exit(EXIT_FAILURE);
}

if( (counter = shmat(shm_id, NULL, 0)) < 0 ){
  perror("shmat");
  _exit(EXIT_FAILURE);
}


fprintf(stdout, "done\n%d: creating semaphore... ", getpid());

sem_t *sem;
unsigned int value = 1;

sem = sem_open(SEM, O_CREAT, 0666, value);

if( sem_init(sem, 1, 1) < 0){
  perror("sem_init");
  _exit(EXIT_FAILURE);
}

fprintf(stdout, "done\n%d: spawing %d child processes... ", getpid(),
 CHILDREN);

pid_t cpid;
for(int i = 0; i < CHILDREN; i++){
  if( (cpid = fork()) < 0){
    sem_unlink(SEM);
    sem_close(sem);
    perror("fork");
    _exit(EXIT_FAILURE);
  } else if( cpid == 0 ){
    for(int j = 0; j < MAX; j++){
      sem_wait(sem);
      fprintf(stdout, "\t%d: entering critical region... ", getpid());
      ++*counter;
      fprintf(stdout, "leaving now\n");
      sem_post(sem);
    }
    _exit(EXIT_SUCCESS);
  }
}

fprintf(stdout, "done\n%d: waiting for children... ", getpid());

while( wait(NULL) > 0);

if( sem_unlink(SEM) < 0){
  perror("sem_unlink");
  return EXIT_FAILURE;
}

if( sem_close(sem) < 0){
```

```
    perror("sem_close");
    return EXIT_FAILURE;
  }

  fprintf(stdout, "done\n%d: printing result: %d\n", getpid(), *counter);

  fprintf(stdout, "%d: writing to pipe... ", getpid());
  FILE* fifo_fp;

  if( (fifo_fp = fopen(FIFO, "w")) == NULL ){
    perror("fopen");
    _exit(EXIT_FAILURE);
  }

  fprintf(fifo_fp, "%d", *counter);

  if( fclose(fifo_fp) < 0 ){
    perror("fclose");
    return EXIT_FAILURE;
  }

  fprintf(stdout, "done\n");

  return EXIT_SUCCESS;
}
```