Seminar 8: Interaction of processes through shared memory

Processes isolate programs from each other; each of them gets its own virtual address space and knows nothing about other programs. However, processes sometimes need to communicate. The set of tools for communication between processes is quite wide:

- Shared files
- Shares memory
- Pipes, named and unnamed
- Message queues
- Sockets
- Signals

Today we will take a look at shared memory. It can also be used via `mmap`.

Fork (10 minutes)

First of all, you need to understand the mechanism for creating new processes. This is done using the `fork` system call, which creates an exact copy of the process. Then, instead of a copy, we can replace the application image by calling `execve`.

If we just want to make a copy of the parent process that behaves a little differently, `execve` is not even needed.

Question: parse this source file.

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {

// pid_t is, like int, a numeric type for process IDs

    pid_t pid = fork();

// depending on the return value that gets into pid, we will know if we are
in the parent process or in the child.

    switch(pid) {
        case -1:
            perror("fork");
            return -1;
        case 0:
            // Child
            printf("Child: my pid = %i, returned pid = %i\n", getpid(), pid);
            break;
        default:
            // Parent
            printf("Parent: my pid = %i, returned pid = %i\n", getpid(),
pid);
```

```
            break;
    }
    return 0;
}
```

Question 1: what exactly does `fork()` return in the parent process? And what's in the child? See `man fork`.

Question 2: Compile and run the program. What is the output? Place it to the form.

Creating a shared memory (40 minutes)

Now let's look at an example of creating a region of memory that is shared between processes.

Question 3: Study the program. Compile it, run it. Check if the PID of the child process is correct using `pstree` or other ways. Send the screenshot of your terminal (shell).

Question 4: Study the program. Compile it, run it. Examine the contents `of/proc/PID/maps` for the parent and child processes, find the shared memory area. Send the screenshot of your terminal (shell).

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <string.h>
#include <unistd.h>

void* create_shared_memory(size_t size) {
  return mmap(NULL,
              size,
              PROT_READ | PROT_WRITE,
              MAP_SHARED | MAP_ANONYMOUS,
              -1, 0);
}


int main() {
  void* shmem = create_shared_memory(128);

  printf("Shared memory at: %p\n" , shmem);
  int pid = fork();

  if (pid == 0) {
    while (1);
  } else {
    printf("Child's pid is: %d\n", pid);
    while(1);
  }
}
```

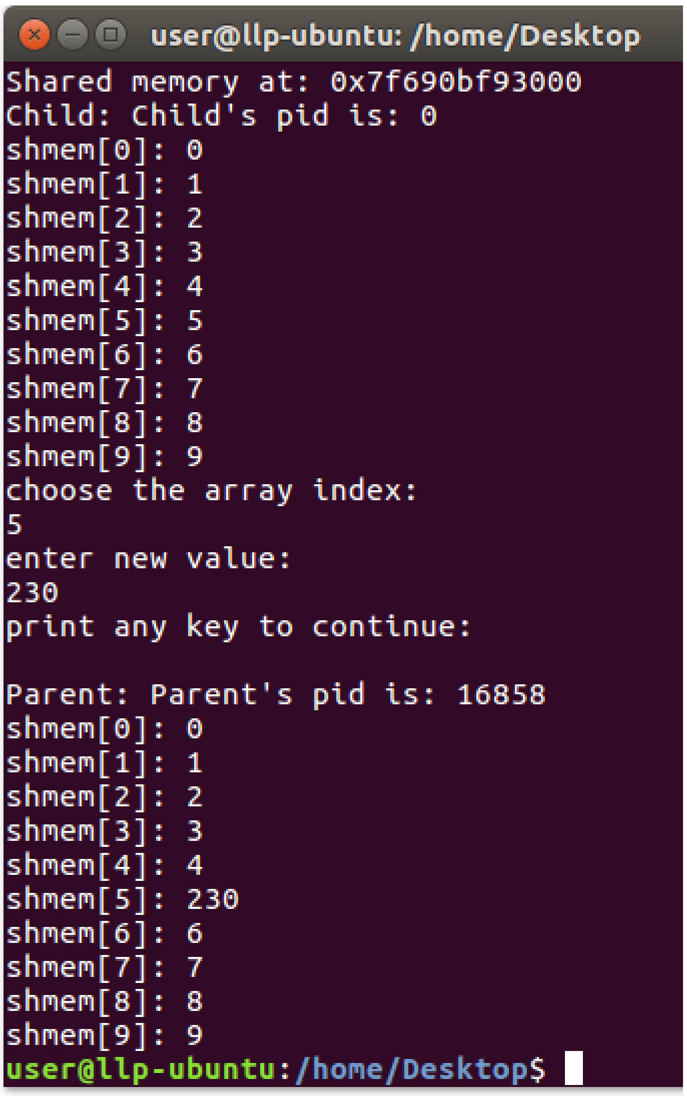
Finally, let's try to write something to this memory! To do this, we need to synchronize the parent and child processes using a mutex. For the sake of simplicity, we will do some code for now, which is WRONG, since it has no synchronization (mutex). However, it will demonstrate the exchange of information between processes through memory.

Question 5. modify the program below as follows and send it through the form:

- It is necessary to allocate memory for 10 numbers of type int and make it common for the process and the child process. Fill in the numbers from 1 to 10. (Already done)
- The parent process will wait for a key press with getchar (). After that, it should display all 10 numbers on the screen.
- The child process reads two numbers using scanf: the array index and the new value. It changes the corresponding number in the array and exits.
- You may need wait(), scanf(), printf(), getchar(), exit().

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

void* create_shared_memory(size_t size) {
  return mmap(NULL,
              size,
              PROT_READ | PROT_WRITE,
              MAP_SHARED | MAP_ANONYMOUS,
              -1, 0);
}

void* print_array(int* array, size_t size){
        for (size_t i = 0; i<size; i++){
        printf("shmem[%zd]: %d\n", i, array[i]);
  }
}



int main() {


  int* shmem = create_shared_memory(10*sizeof(int));

  printf("Shared memory at: %p\n" , shmem);

  for (size_t i = 0; i<10; i++){
        shmem[i]= (int)i;
  }


printf("Shared memory at: %p\n" , shmem);

int pid = fork();




  if (pid == 0) {
    while (1); // place here your code instead

  } else {
    printf("Child's pid is: %d\n", pid);
    while(1); // place here your code instead
  }
}
```

The output your program should give:



```
Shared memory at: 0x7f690bf93000
Child: Child's pid is: 0
shmem[0]: 0
shmem[1]: 1
shmem[2]: 2
shmem[3]: 3
shmem[4]: 4
shmem[5]: 5
shmem[6]: 6
shmem[7]: 7
shmem[8]: 8
shmem[9]: 9
choose the array index:
5
enter new value:
230
print any key to continue:

Parent: Parent's pid is: 16858
shmem[0]: 0
shmem[1]: 1
shmem[2]: 2
shmem[3]: 3
shmem[4]: 4
shmem[5]: 230
shmem[6]: 6
shmem[7]: 7
shmem[8]: 8
shmem[9]: 9
user@llp-ubuntu:/home/Desktop$
```