QUIDET Victor
TARBE Théophile
ING4 Gr01 - SI (Inter)

LAB2 :
Processes & Shared Memory

1 - Shared Memory

lab2.c > main(int, char **)

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>

#define KEY 4567
#define PERMS 0660

int main(int argc, char **argv)
{
  //declaration of variables
  int id;
  int i;
  int *ptr;

  //display segments of shared memory by id on the system
  system("ipcs -m");
  //asking for a shared memory segment with shmget()
  id = shmget(KEY, sizeof(int), IPC_CREAT | PERMS);
  //display segments of shared memory by id on the system
  system("ipcs -m");
  //After a shared memory ID is returned, the next step is to attach it to the address space of a process : shmat()
  ptr = (int *)shmat(id, NULL, 0);

  //initializationof variables
  *ptr = 54;
  i = 54;

  if (fork() == 0) //child process : we increment values of *ptr and i and we display them
  {
    (*ptr)++;
    i++;
    printf("Value of *ptr = %d\nValue of i = %d\n", *ptr, i);
    exit(0);
  }
  else // parent process
  {
    wait(NULL); //will block parent process until any of its children has finished
    printf("Value of *ptr = %d\nValue of i = %d\n", *ptr, i);
    shmctl(id, IPC_RMID, NULL); //shmctl() is used to detach a shared memory
  }
}
```

```
macbook-air-de-theophile-2:lab2 theophiletarbe$ gcc -g -o exe lab2.c
macbook-air-de-theophile-2:lab2 theophiletarbe$ ./exe
IPC status from <running system> as of Fri Sep 18 15:00:52 CEST 2020
T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:

IPC status from <running system> as of Fri Sep 18 15:00:52 CEST 2020
T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m 131072 0x000011d7 --rw-rw---- theophiletarbe    staff

Value of *ptr = 55
Value of i = 55
Value of *ptr = 55
Value of i = 54
```

1. What could you infer from the output regarding the state of i and *ptr?

We can observe that :

- For the Child Process : `*ptr` & `id` = 55 because they are incremented into the loop "`if (fork() == 0)`"
- For the Parent Process : `*ptr = 55` and `id = 54`. In fact, the function `shmat()` attach a segment of memory between the child and the parent for `*ptr`. As a result, `*ptr` is also incremented in the parent process.

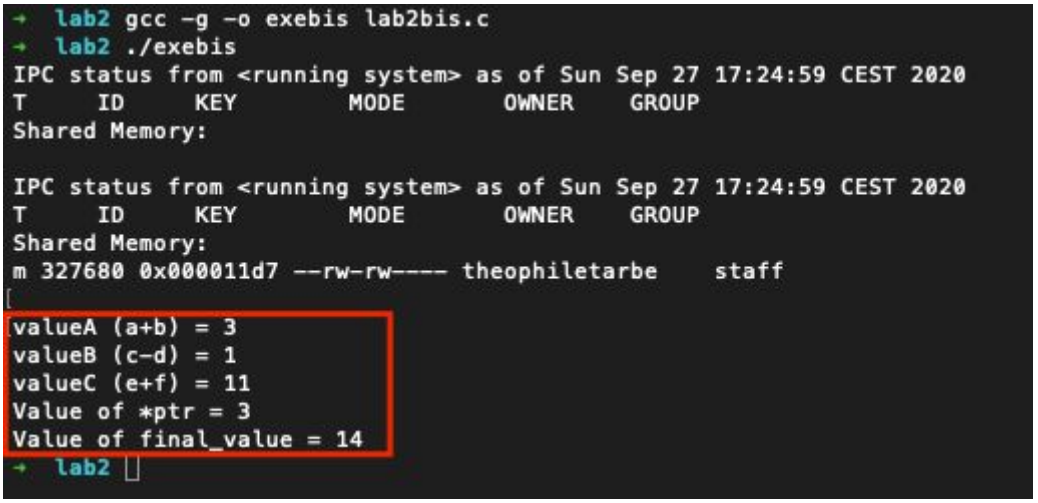
2. Comments : they are already on the screenshot of the code above.

2 - Parallel Computing

Write a program that computes the following expression “(a + b) * (c - d) + (e + f)” using 3 different processes.

lab2bis.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>

#define KEY 4567
#define PERMS 0660

int main(int argc, char **argv)
{
  //declaration of variables
  int id;
  int *ptr;

  int a,b,c,d,e,f;
  int valueA, valueB, valueC, final_value;

  //display segments of shared memory by id on the system
  system("ipcs -m");
  //asking for a shared memory segment with shmget()
  id = shmget(KEY, sizeof(int), IPC_CREAT | PERMS);
  //display segments of shared memory by id on the system
  system("ipcs -m");
  //After a shared memory ID is returned, the next step is to attach it to the address space of a process : shmat()
  ptr = (int *)shmat(id, NULL, 0);

  //initializationof variables
  *ptr = 0;
  a = 1;
  b = 2;
  c = 4;
  d = 3;
  e = 5;
  f = 6;

  if (fork() == 0) //child process
  {
    valueA = (a+b); // 1st calcul
    printf("valueA (a+b) = %d\n", valueA);

    if (fork() == 0)
    {
      valueB = (c-d); //2nd calcul
      printf("valueB (c-d) = %d\n", valueB);
    }
    *ptr = (valueA * valueB); //*ptr takes the final result of the cild preocess
    //printf("Value of *ptr = %d\n", *ptr);
    exit(0);
  }
  else // parent process
  {
    wait(NULL); //will block parent process until any of its children has finished
    valueC = (e+f); //3rd calcul
    printf("valueC (e+f) = %d\n", valueC);
    printf("Value of *ptr = %d\n", *ptr);
    final_value = (*ptr + valueC); //we use the result of *ptr with the 3rd calcul for the final value
    printf("Value of final_value = %d\n", final_value);
    shmctl(id, IPC_RMID, NULL); //shmctl() is used to detach a shared memory
  }
}
```



On this screenshot we can observe different results :

- The variable `valueA` is given by the 1st Child process which computes `(a+b)` and stores the result.
- The variable `valueB` is given by the 2nd Child process which computes `(c+d)` and stores the result.
- Then, thanks to the function `shmat()`, wich attach a segment of memory between the child and the parent for the variable `*ptr`, we can store the final value of the 2 Child processes into the variable `*ptr`.
- Finally, the Parent process (3rd process), computes the `valueC`, and as it shares the value of `*ptr` with the Child process, it can calculate the final result of the operation `final_value` with a simple operation.

The code and the results on the Terminal display how to make a complex operation with 3 different processes.

Additional question:

If we do the same evaluation but this time instead of using "wait" between processes, we use shared flags to announce the end of a process. What is the difference between the two versions ?

The `wait()` system call suspends execution of the calling process until one of its children terminates. Without the `wait()` it would be the contrary. And “shared flags” (or lock) signal whether a process can or cannot enter the critical section. Therefore, the flag has only two values :

- 0 (critical section is unlocked)
- 1 (critical section is locked).

When a process wants to enter the critical section it checks to see if the flag's value is 0. If it is, the process enters and sets the flag to 1. Otherwise, if the flag is already 1, that means that another process is currently interacting with the variables and files in the critical section, and so the process waits until the flag changes to 0.

Sources :

https://www.geeksforgeeks.org/fork-memory-shared-bw-processes-created-using/?ref=rp

https://www.geeksforgeeks.org/fork-system-call/

https://www.geeksforgeeks.org/calculation-parent-child-process-using-fork/?ref=rp

https://www.geeksforgeeks.org/creating-multiple-process-using-fork/?ref=rp

https://www.geeksforgeeks.org/fork-memory-shared-bw-processes-created-using/?ref=rp

https://www.geeksforgeeks.org/factorial-calculation-using-fork-c-linux/?ref=rp

http://www.cyberiapc.com/os/busywait-sharedflag.htm