POLITÉCNICO DE PORTALEGRE

Escola Superior de Tecnologia, Gestão e Design

OPERATING SYSTEMS

Exercise 4 – Shared Memory

1. Create a Shared Memory

To create a shared memory, use the `shmget` function. For example:

```
shmget(1000, 800, 0600 | IPC_CREAT)
```

The first and last arguments are similar to those of the other "`get`" functions. The second argument indicates the size (in number of bytes) of the memory to be created.

The `ipcs` command allows you to view the list of characteristics of the memory resources created, namely their size.

2. Associate a Pointer to the Created Shared Memory

The memory created (or the memory that the process associates with) with `shmget` is an operating system resource that is only usable once it is *mapped* into the process's address space. This mapping is done through a call to the `shmat` function (from "attach"). For example:

```
int *ptr_mem;
ptr_mem = (int *)shmat(shm_id, NULL, 0);
```

`shmat` function which returns, in the `ptr_mem` variable, a pointer to the shared memory. In other words, after this call we can access the shared memory through the `ptr_mem` pointer. This pointer is of type `void *`. Although it physically "points" to memory, for compilation purposes it is only usable once it has been typed.

Let's assume, for example, that the memory will contain a set of characters. Then, you cast the original `void*` pointer to this data type, that is, to `char *`.
For example:

```
char *ptr;
ptr = (char *)ptr_mem;
```

Example 2.a)

The following example creates a shared memory and initializes it by filling all positions with the '−' character.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }
#define exit_on_null(s,m) if ( s == NULL ) { perror(m); exit(1); }

int main(int argc, char *argv[])
{
   int shm_id, status;
   char *ptr;
   int i;

   shm_id = shmget(1000, 800, 0600 | IPC_CREAT);
   exit_on_error(shm_id, "Creation");

   ptr = (char *)shmat(shm_id, NULL, 0);
   exit_on_null(ptr, "Attach");

   for (i = 0; i < 800; i++)
      ptr[i] = '-';
}
```

Example 2.b)

The following example reads shared memory and displays its contents on the screen.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }
#define exit_on_null(s,m) if ( s == NULL ) { perror(m); exit(1); }

int main(int argc, char *argv[])
{
    int shm_id, status;
    char *ptr;
    int i;

    shm_id = shmget(1000, 800, 0600 | IPC_CREAT);
    exit_on_error(shm_id, "Creation");

    ptr = shmat(shm_id, NULL, 0);
    exit_on_null(ptr, "Attach");

printf("12345678901234567890123456789012345678901234567890123456789012345678
901234567890\n");
    for (i = 0; i < 800; i++) {
        printf ("%c", ptr[i]);
        if ((i+1) % 80 == 0)
            printf ("\n");
```

```
    }
printf("12345678901234567890123456789012345678901234567890123456789012345678
901234567890\n");
}
```

3. Data Structuring

In principle, shared memory is just a space where a set of bytes can be stored. It is up to the programmer to structure this space in order to place the data he needs to manipulate in it.

The following example structures memory as a row data array. Tthis line type is, in this case, a 40-character string, but it could equally be a structure or a class.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }
#define exit_on_null(s,m) if ( s == NULL ) { perror(m); exit(1); }

typedef char line[20];

int main(int argc, char *argv[])
{
   int shm_id, status;
   line *ptr;
   int i;

   shm_id = shmget(1000, 800, 0600 | IPC_CREAT);
   exit_on_error(shm_id, "Creation");

   ptr = (line *)shmat(shm_id, NULL, 0);
   exit_on_null(ptr, "Attach");

   for (i = 0; i < 20; i++) {
      sprintf(ptr[i], "line number %d", i+1);
   }
}
```

Try this program and try to see the memory with the program in example **2.b).**
Execute, in order, the program in example **2.a)** and then this program. See the final result with the program in example **2.b)** and explain. Does it feel like there's one less character in each line?

4. Simultaneous Accesses

Nothing prevents two or more processes from accessing shared memory simultaneously. The following example continuously writes a character to shared memory. Make two or more of these programs run simultaneously and see the memory evolution with the program in example 4.b) that shows the memory, continuously, every 1 second.

Example 4.a) Continuous Writing in Memory

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }
#define exit_on_null(s,m) if ( s == NULL ) { perror(m); exit(1); }

int main(int argc, char *argv[])
{
   int shm_id, status;
   char *ptr;
   int i, k;
   int c;

   shm_id = shmget(1000, 800, 0600 | IPC_CREAT);
   exit_on_error(shm_id, "Creation");

   ptr = (char *)shmat(shm_id, NULL, 0);
   exit_on_null(ptr, "Attach");
   c = 'A' + getpid() % 26;
   printf("Writing '%c'\n");
   while (1) {
      for (i = 0; i < 800; i++) {
         ptr[i] = c;
            for (k = 0; k < 1000000; k++)
               k = k * 2.0 / 2.0;
      }
   }
}
```

Example 4.b) Memory Content Monitor

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }
#define exit_on_null(s,m) if ( s == NULL ) { perror(m); exit(1); }

int main(int argc, char *argv[])
{
    int shm_id, status;
    char *ptr;
    int i;

    shm_id = shmget(1000, 800, 0600 | IPC_CREAT);
    exit_on_error(shm_id, "Creation");

    ptr = (char *)shmat(shm_id, NULL, 0);
    exit_on_null(ptr, "Attach");
    while (1) {
printf("12345678901234567890123456789012345678901234567890123456789012345678
901234567890\n");
        for (i = 0; i < 800; i++) {
```

```
            printf("%c", ptr[i]);
            if ((i+1) % 40 == 0)
                printf ("\n");
        }
printf("12345678901234567890123456789012345678901234567890123456789012345678
901234567890\n");
        sleep(1);
   }
}
```

5. Exclusion

Simultaneous access to memory by multiple processes can rarely be permitted without rules. It is up to the programmer to define and implement these rules, usually using exclusion mechanisms using semaphores.

TODO: Access With Exclusion Semaphore (mutex)

Study the code provided in PAE (semaphore.c and semaphore.h) and make necessary changes to the previous programs to ensure mutual exclusion to shared memory.

Try running multiple programs 4. a) simultaneously and monitor the memory contents with the program 4. b). It should be clear that each process now completely fills the memory.