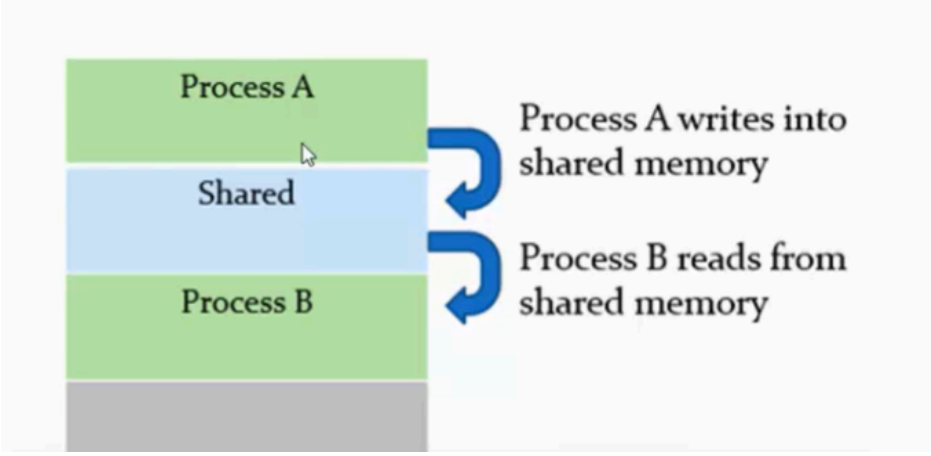
Reading Material: Shared Memory

In the previous lab, we used the exit status of a terminating child to return the calculated values from the child to the parent. But if we want two independent processes to communicate more than that we would have to use shared memory and message passing.

Shared Memory

We can create a shared memory segment if we want the data produced by one process to be accessible by another process.



Shared memory segments have a unique number identifier which allows other processes to "attach" themselves to a specific shared memory segment.
The system calls used to create, attach, detach, and destroy a shared memory segment are:

shmget()

```
shmget(key_t key, size_t size, int shmflg)
```

shmget() returns the identifier of the shared memory segment associated with the value of the argument key. It may be used to obtain the identifier of a previously created shared memory segment (when shmflg is zero and the key does not have the value IPC_PRIVATE) or create a new shared memory segment.
Key: Unique identifier of the segment
Size: the size of the shared segment to be created (is rounded up to a multiple of PAGE_SIZE)
Shmflg: Is used to specify whether the segment is to be created and/or the type of access (read/write) the calling process will have.

The call to create a new shared memory segment can be made this way:

```
shmid = shmget(key, 1000, SHM_R | SHM_W | IPC_CREAT);
//adding the IPC_CREAT flag to indicate that the shared memory needs to be
created. SHM_R, SHM_W flags set to allow read and write operations
if (shmid < 0) {
     perror("shmget");
     exit(1);
}
```

A process that wants to access an already created shared memory segment must not specify IPC_CREAT in the shmflg argument.

shmat()

```
shmat(int shmid, const void *shmaddr, int shmflg)
```

shmat() is used to attach the shared memory segment identified by shmid to the address space of the calling process. After successfully attaching itself, a pointer to the segment is returned and the calling process can access the shared memory as it would through any other pointer.

Shmid: Identifier returned by the shmget call for the shared segment the process want to attach.
Shmaddr: The address at which we want to attach the segment. We will be setting this parameter NULL. This allows the OS to select an address to which the shared memory is attached.
Shmflg: Specifies more information about the access permissions and/or the address at which the segment is to be attached. Can be set to 0 for our use.

```
shm = (char *)shmat(shmid, NULL, 0);
// using shmat to attach to the shared memory block. Returns pointer to
shm that can be used to access it.
if (shm == (char *)-1)
{
   perror("shmat");
   exit(1);
}
```

shmdt()

```
shmdt(const void *shmaddr)
```

Detaches the shared memory segment pointed to by shmaddr. Shmaddr must be a valid address returned by a shmat() call made before.

shmctl()

```
shmctl(int shmid, int cmd, struct shmid_ds *buf)
```

shmctl() performs the operation specified by the cmd argument. But we will mainly be using it to destroy the shared memory segment identified by shmid.
Shmid: Identifier returned by call to shmget() while creating shared memory segment
Cmd: Command which specifies the operation to be performed. We will be setting it to IPC_RMID. This marks the segment for destruction. The caller must be the creator/owner of the segment.
Buf: Pointer to a shmid_ds struct which is used to store information about the shared segment. We will be setting it to NULL as we only want to destroy the shared memory segment.

```
if((shmctl(shmid, IPC_RMID, NULL)) == -1) {
   perror("shmctl");
   exit(1);
}
```

Note: Use man pages if you want to read about these system calls in detail

Another system call that might be useful in your labs (Use this if a process is waiting for something or if you want to suspend execution)

usleep()

```
usleep(__useconds_t useconds)
```

Suspends execution for useconds amount of microseconds
Example:

```
while(buf[0] != 5) {
  usleep((rand() % 1000) + 1000);
}
```

In the above example, the line of code can be used to suspend the execution of a process for a random amount of microseconds between 1000 and 1999 till that variable is equal to 5