Assignment 2

Satvik Chauhan (Y9521),Pankaj More (Y9402), Diwakar chauhan(Y9203), Lakshya Khurana (Y930

Shared Memory

Four system calls are implemented for implementing and managing shared memory :

- int SharedMemoryOpen(int size) : It creates a memory of "pageSize" and checks whether walue of size is less than pageSize or not . To implement this, when a process is created an extra page is added to it's address space. This system call adds pagetable entry for that extra memory part, thus assigning a physical page for that virtual address-space. When called for the first time it just returns the starting address of the added physical memory. When called later on it makes that memory shared to the current thread.
- void SharedMemoryClose(int size) : It closes the shared memory for the current thread. It check whether the current thread has shared momory or not, it it has then the system call closes that.
- int SharedMemoryWrite(int mem, int size, int val) : This system call writes "val" of size "size" at the memory location "mem" of the shared memory. Ths value of mem is given with respect to one single page i.e. it renges frome 0 to 1023. It returns the next unoccupied memory location in the shared memory.
- int SharedMemoryRead(int mem, int size) : It reads the value of size "size" from the memory location "mem" and returns it's value. "mem" is similar to that in the previous system call.
- Shared Memory Test Program : shmtest.c

Semaphores

Semaphores are implemented using the default semaphore already implemented in synch.h and synch.cc files. The following system calls are implemented :

- int SCreate(int value) : Takes a value and creates the semaphore returning the id of the created semaphore which can be used in the rest of the system calls.
- SWait(int id) : Performs a semaphore wait call and decrease the value by 1. Sleeps the process if value becomes less than 1.
- SSignal(int id) : Performs a semaphore signal call and increase the value by 1. Wakes up the process if value becomes greater than 0.
- SDestroy(int id) : Destroys the semaphore.

Dining philosopher problem

- **Deadlock :** We have ensured that at any time only 4 philosophers are allowed at the table using a semaphore with value 4. This prevents deadlock .
- **Mutual Exclusion :** We have 5 semaphores for each chopstick and before eating any philosopher checks whether its left and right chopsticks are available or not. This ensures mutual exclusion.
- **Starvation :** Starvation is prevented because a FCFS queue is maintained for a process waiting on any semaphore. So a process which performs a wait call first will be woken before a process which performed a wait call afterwards. Now, the only way a process can starve is due to scheduler not giving fair chance to all the processes (a process represent a philosopher) . To prevent this we have assigned the same priorities to all the processes. The processes having same priorities are served in FCFS which ensures no starvation.
- **Empirical Result:**
  Program to run : assignment1.c
  Output of the Dining Philosophore for 1001 eatings is : 201 201 196 203 200
  Since all the five philosophers(processes) eat approximately the same number of times which proves that there is no starvation.