Conceptual Questions

1. What is fork(), how does it differ from multi-threading?

fork() is a function that invokes a system call to duplicate the current process. This function takes no arguments and returns the process ID of the new process. The newly created process will be a child process of the process that duplicated itself. Once duplicated, both the parent and child processes will continue to execute the code below the fork instruction.

2. What is IPC? Describe methods of performing IPC?

IPC (inter process communication) is a mechanism which allows processes to communicate with each other and synchronize their actions. The two methods of using IPC are shared memory and message passing.

Shared memory

Two processes share a common space or memory location where one process produce and store data and another processe can consume it.

Message passing

This method of communication does not require any kind of shared memory. Instead, two processes will establish a communication link and start exchanging messages using send() and recieve() primitives.

3. Explain semaphores, how they work and how they differ from mutex?

Semaphores are a mechanism that allow n-number of threads/processes to access a shared resource. They provide atomic operations, which are operations that cannot be interrupted, to control the access and modification of data. The semaphore contains a predetermined number, which represents the number of processes using the critical section. If the semaphore value was initally 3, then it can allow up to 3 processes to lock it until it reaches zero. The main difference between semaphores and mutexes are that semaphores are signal based and allow multiple threads to access a critical section while mutexes are owned by threads and only allow 1 thread to access the critical section at a time.

4. Explain wait(P) and signal(V).

Wait and signal are atomic operations that are used to interact with the integer value in the semaphore object. The wait function will attempt to decrement the semaphore value, unless it is negative. At this point, the thread that called the wait function will block until the semaphore value becomes positive again. The signal function will increment the semaphore value. At this point, if the

semaphore value is zero, it will use some scheduling method to allow the next thread in line to return from it's wait call. If the value is positive, then no threads are currently in line.

5. Explain the functions in semaphore.h

int sem_init(sem_t *sem, int pshared, unsigned int value);

Initializes the semaphore referenced by *sem* with the initialized value of *value*.

int sem_destroy(sem_t *sem);

Destroys the semaphore referenced by *sem*, freeing it's resources. Only a semaphore initialized with sem_init can be destroyed with this function.

int sem_wait(sem_t *sem);

Locks the semaphore referenced by *sem* by decrementing its value. If the semaphore value is currently zero, the calling thread will not return until it either locks the semaphore or is interupted by a signal.

int sem_post(sem_t *sem);

Unlocks the semaphore referenced by *sem* by incrementing its value. If the semaphore value is postive after this, then the next thread in line can lock the semaphore.

int sem_trywait(sem_t *sem);

Locks the semaphore referenced by *sem* only if it is currently unlocked, that is if the value is currently positive.

Application Questions

All code and output can be found here.

Link to clone repository: git@github.com:AshwinK97/Operating-Systems.git