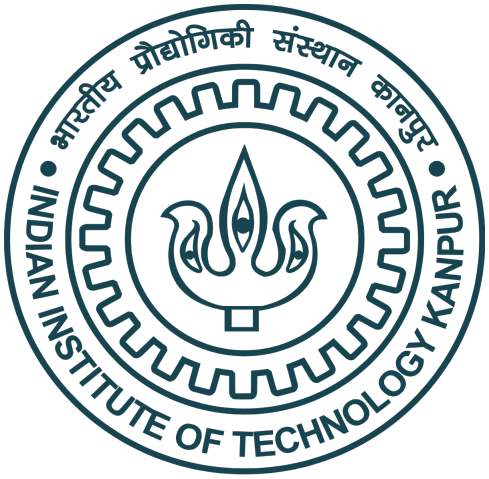
CS330A: Operating Systems



Assignment 3

Akhil Agrawal
200076

Akshat Garg
200084

Suket Raj
201013

Uttam Kumar
201071

November 15, 2022

1 Implementation of Condition Variable

The condition variable of type *cond_t* is implemented as a structure in *kernel/condvar.h* containing a single integer attribute *value* which will take either 0 or 1.

2 Implementation of Semaphore

Semaphore structure is implemented in *kernel/semaphore.h* as a structure *semaphore* with its value x and a sleeplock lk as its attributes.
In the file *kernel/semaphore.c* the three functions *sem_init, sem_wait, sem_post* are implemented which initialize the value of x in the structure semaphore, decrement the value by 1 and increment the value by 1 respectively. The sleeplock is held while doing the changes to the value of x. If the value of x goes negative in *sem_wait*, the process is sent to sleep.

3 Implementation of System Calls

Structure of barrier:
The barrier has the following attributes:

1. *flag:* keeps a flag on whether the barrier has been allocated.
2. *num_processes:* number of processes to be held by the barrier
3. *instance:* Number of instances observed yet
4. *cv:* condition variable
5. *lk:* the sleeplock

3.1 barrier_alloc

It iterates through the barrier array and finds an unallocated barrier by its flag and return its index as the barrier index.

3.2 barrier

Checks for the number of processes received yet, if it has to be barrierred here, it is sent to *condsleep*. Otherwise the condition value is changed and broadcasted and the processes are allowed to pass.

3.3 barrier_free

It resets the barrier to the initial position, i.e. flag is set to 0 again.

Structure of bounded buffer condition :
The struture bounded_buffer_condition contains the following attributes:

1. *head:* The next consumed item index
2. *tail:* The next produced item index

3. *buffer:* An array of structure buffer_elem which contains:
    - *x:* The value of the buffer element
    - *full:* Boolean to check empty or not
    - *lock:* sleeplock
    - *deleted:* condition variable
    - *inserted:* condition variable
4. *lock_delete:* sleeplock
5. *lock_insert:* sleeplock
6. *lock_print:* sleeplock

3.4 buffer_cond_init

Initialises the bounded buffer condition i.e. the conditions variables are initialsed and the next producer and next consumer both are set to point to 0.

3.5 cond_produce

The idea is to set buffer value at the next producer index to the value provided. Then cyclically increment the next producer.

3.6 cond_consume

The idea is to get the value stored in the buffer at the next consumer index and set the index buffer to be available for future produces. Again cyclically increment the next consumer.

Structure of bounded buffer semaphore:
The struture bounded_buffer_semaphore contains the following attributes:

1. *size:* The size of the buffer
2. *empty:* semaphore
3. *full:* semaphore
4. *prod:* semaphore
5. *cons:* semaphore
6. *buffer:* An array of fixed size
7. *nextp:* The next produced item index
8. *nextc:* The next consumed item index

3.7 buffer_sem_init

Initialises the bounded buffer semaphore i.e. the size is set to 20, the semaphores are initialsed and the next producer and next consumer both are set to 0.

3.8 sem_produce

The idea is to set buffer value at the next producer index to the value provided. Then cyclically increment the next producer.

3.9 sem_consume

The idea is to get the value stored in the buffer at the next consumer index and set the index buffer to be available for future produces. Again cyclically increment the next consumer.

4 Observations

It was observed that the program implementation of bounded buffer problem through condition vairables that is condprodconstest was faster than the implementation through semaphores that is semprodconstest. The programs were tested on various inputs covering corner cases like single producer or single consumer to obtain the result that semaphore implementation was slower than the condition variable one. The result can be explained as the semaphore has a heavier and complicated structure than the condition variable processing which takes more time.