Operating Systems Lab: CS39002

Assignment 5: Implementation of multiple producer-consumer system where producers create prioritized jobs

Group 31: Report

Name: Abhinav Bohra **Roll No:** 18CS30049 **Email ID:** abhinavbohra@iitkgp.ac.in

Name: Animesh Jain **Roll No:** 18CS10004 **Email ID:** animesh99@iitkgp.ac.in

Part 1: Implement a producer / consumer set of *processes* using shared memory

Data Structures

```
typedef struct Job {
    int producer_pid;           // Producer process_id
    int producer_no;            // Producer Number
    int priority;               // Priority between 1 and 10
    int compute_time;           // Compute Time between 1 and 4
    int job_id;                 // Job ID between 1 and 100000
} JOB;


struct priority_queue {
    JOB job_queue[QUEUE_SIZE];       // Queue array
    int back;                        // Last index of Queue;
};


typedef struct SHMSegment {
    struct priority_queue job_queue;  //priority queue of 8 elements
    int job_created;                  //counter of number of jobs created
    int job_completed;                //counter of number of jobs completed
} SMT;
```

Algorithms and User-Defined Functions

We have divided the code in the following 5 parts to maintain modularity and enhance understandability of the code: -

Main	Main Function (forks multiple producers and consumers)
Producer	Producer Function (Creates and Inserts jobs)
Consumer	Consumer Function (Removes jobs)
Queue	Functions for Priority Queue implementation
Shared	Functions related to semaphores and shared memory

1. Main Function

- Takes number of producers, consumers and jobs as user-input from terminal
- We maintain an array ‘all_pid” to store ids of all the processes (producer as well as consumer)
- Creates a shared memory segment SHM, which is shared among all the producer and consumer processes
- Forks every producer process and calls producer_main(..) in child process (pid==0)
- Forks every consumer process and calls consumer_main(..) in child process (pid==0)
- Wait till both job_created counter and job_completed counter reaches a specified number of jobs
- Kills the parent and child process using SIGTERM
- Clears shared memory segment
- Calculates and prints total execution time of program

2. Producer

Function -> int insert_job(JOB job, SMT *shmseg)

- Takes shared memory segment and new job as input
- Returns -1 if queue is full
- Inserts job in queue (sorted w.r.t priority) if space is available
- Enqueue function implements this by finding the right place for insertion & shifting other elements of the queue accordingly
- Prints Job Details as mentioned in assignment after successful addition of job to queue

Function -> JOB produce_job(int producer_no)

- Creates an element of type struct JOB
- Populates the elements of structure as per assignment guidelines

Function -> int producer_main(int i, int NJ)

- Each producer process generates a computing job,
- Waits for a random interval of time between 0 and 3 seconds
- Locks the shared memory segment using semaphore before insertion
- Inserts the computing job in shared memory queue, if space available
- While insertion, if queue is full, it releases the lock and waits till consumer process consumes a job and creates an empty space in the queue of shared memory
- After successful insertion, the producer releases the lock & repeats the process.

3. Consumer

Function -> JOB remove_job(int consumer_no, SMT* shmseg)

- Takes shared memory segment and consumer number as input
- Returns a dummy job with job_id = -1, if queue is empty
- Retrieves the highest priority job if queue is not empty
- Removes the job from the queue by using deque function
- Prints details of the retrieved job as mentioned in the assignment
- Returns the retrieved job

Function -> int consumer_main(int i, int NJ)

- Each consumer process waits for a random interval of time between 0 and 3 seconds
- Locks the shared memory segment using semaphore before job retrieval
- Retrieves the job with highest priority in the shared memory

- While retrieval, if queue is empty, it releases the lock and waits till producer process adds a job to the queue
- Then increases the job_completed counter
- Sleeps for "compute time" seconds of the retrieved job

4. Queue

We have implemented a ***priority queue*** using the following functions: -

- void init_queue(struct priority_queue *pq);   //Initialises queue
- int isEmpty(struct priority_queue pq);   //Check if queue is empty
- int isFull(struct priority_queue pq);   //Check if queue is full
- int enqueue(struct priority_queue *pq, JOB job);   //Add element to queue
- JOB dequeue(struct priority_queue *pq);   //Remove element from queue

5. Shared Memory

To avoid race condition, we have used ***semaphores*** to lock the critical section of program, i.e., whenever the shared memory segment is accessed by a process.

Function -> int init_SHM(SMT* shmseg);

- Initialises Shared Memory Segment varaibles as
- shmseg->job_completed = 0;
- shmseg->job_created = 0;
- init_queue(&shmseg->pq);

Function -> SMT *create_SHM(int *shm_id);

- Takes shm_id as input and creates shared memory segment using shmget()
- Flags possible error while creating/ attaching segment.
- Returns a pointer to a shared memory buffer that the producer can write to.

Function -> int create_semaphore_set();

- Creates FULL and EMPTY semaphores

Part 2: Implement a producer / consumer set of ***threads*** using shared memory

Data Structures

**** SAME AS PART 1 ****

Algorithms and User-Defined Functions

1. Main Function

- Takes number of producers, consumers and jobs as user-input from terminal
- Uses a global memory segment shm, as in the case of threads, the memory is already shared as ***threads share address space***.
- Producer threads producer[*i*] are created and they call producer_main(..) upon creation
- Consumer threads consumer[*i*] are created and they call consumer_main(..) upon creation
- Wait till both job_created counter and job_completed counter reaches a specified number of jobs
- Kills the producer and consumer threads using pthread_detach(..)
- Calculates and prints total execution time of program

2. Producer

Function -> int insert_job(JOB job, SMT *shmseg)

**** SAME AS PART 1 ****

Function -> JOB produce_job(int producer_no)

**** SAME AS PART 1 ****

Function -> void* producer_main(void* argv)

- The arguments int i, int NJ a recovered from void* argv
- Everything else is same as in Part 1

3. Consumer

Function -> JOB remove_job(int consumer_no, SMT* shmseg)

**** SAME AS PART 1 ****

Function -> void* consumer_main(void* argv)

- The arguments int i, int NJ a recovered from void* argv
- Everything else is same as in Part 1

4. Queue

**** SAME AS PART 1 ****

5. Shared Memory

- The function SMT* create_SHM(int* shm_id) is removed, since there is no need to explicitly share memory between threads as they share common address space
- Global memory segment SMT shm is created
- Everything else is same as in Part 1