Varun Deliwala
Operating Systems End Semester Lab Examination Section 2
AU1940034

1. Print the following Pattern
   A 1 a B 2 b C 3 c … Y 25 y Z 26 z
   Using any one of the following concepts
   a. Multiprocesses (Hint: using 3 child processes)
   b. Multithreads (Hint: using 3 Threads)
   (PS: Process Synchronization or Thread Synchronization is key thing for pattern printing) **[25]**

2. Describe and implement any one of the following **[20]**
   a. Describe the RoundRobin (RR) and Modified RoundRobin (MRR) Algorithm. Also mention the difference between the results of both the algorithms and implement them both in C.
   Reference: https://ieeexplore.ieee.org/document/8392238
   b. Describe the Buddy's Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.

3. [Bonus] Describe what is Producer Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C. **[10]**

Question 1

Using Multithread

In this question, I am using 3 different threads and then we use the Thread_id variable to manage between the threads and then at the same time, I am trying to print the values with the help of a flag variable name fg. I am using 3 other variables to maintain all the upper, lower and the numbers that are printed here. For the printing, 3 different variables are used for printing, named, UpperCs, LowerCs and num.

```
varun@DESKTOP-IJPEUEO:~$ gcc -pthread question_1.c
varun@DESKTOP-IJPEUEO:~$ ./a.out
A 1 a B 2 b C 3 c D 4 d E 5 e F 6 f G 7 g H 8 h I 9 i J 10 j K 11 k L 12 l M 13 m N 14 n O 15 o P 16 p Q 17 q R 18 r S 19 s T 20 t U 21 u V 22 v W 23 w X 24 x Y 25 y Z 26 z
varun@DESKTOP-IJPEUEO:~$ cd
```

Question 2 B

A memory allocation technique known as the buddy memory allocation approach divides memory into divisions in order to deliver the best possible answer to a memory request. In order to discover the best match, this system splits memory in half. Buddy systems come in a variety of shapes and sizes; the simplest and most common are those in which each block is divided into two smaller blocks. Every memory block in this system has an order, which is a number ranging from 0 to a predetermined upper limit. Because the size of a block of order n is proportional to 2n, the blocks are exactly twice the size of blocks of lower order. With power-of-two block sizes, address computation is simple since all buddies are aligned on memory address boundaries that are powers of two. When a larger block is divided, two smaller blocks are created, each of which becomes a unique buddy for the other. Only its one-of-a-kind buddy block can be paired with a split block to rebuild the larger block from which it was separated. To begin, the smallest possible block is determined, i.e. the smallest memory block that can be allocated. If there was no lower limit at all, the system would have a lot of memory and computational overhead keeping track of which regions of memory are allocated and unallocated (e.g., bit-sized allocations were feasible). A low limit, on the other hand, may be preferred to decrease average memory waste per allocation (for allocations that are not multiples of the smallest block in size). The lower limit is usually small enough to decrease average wasted space per allocation while yet being large enough to avoid excessive overhead. The programmer must then pick, or write code to accomplish, the largest feasible order that will fit in the available memory. If the total accessible memory of a computer system is not a power-of-two multiple of the minimum block size, the largest block size may not encompass the whole memory. If the system had 2000 K of physical memory and the order-0 block size was 4 K, the top order constraint would be 8, because an order-8 block (256 order-0 blocks, 1024 K) is the largest block that will fit in memory. As a result, allocating the entire physical memory in one chunk is impossible; the remaining 976 K must be allocated in smaller parts.

Initialization

-> Size of vector of pairs

-> Global vector of pairs to track all the free nodes of various sizes

-> Map used as hash map to store the starting address as key and size of allocated segment key as value

---

We will have to create a function named Buddy, a function to allocate and a function to deallocate

-> The allocate function will the first check the if the bock is available.

> if its available the it will assign and remove it from the free list. Then we will change the starting address of the map with size to make the deallocating easy.
>
> If its not available then we will search for a larger block and then request. if no such memory block is found then we will fail to allocate the memory. If its found then we

will split it to halves. We will push them into the free list and remove the first free block to further split.

-> The deallocate function will first check if the starting address is available or not.

We will have to check if the size of the block stated is available or not. After this we will calculate the buddy number and then search in the free list if the buddy number is free or not. If it is free then we will merge the two buddies to make them a larger block of free memory.

```
varun@DESKTOP-IJPEUEO:~$ g++ question_2B.cpp -o answer -lpthread
varun@DESKTOP-IJPEUEO:~$
varun@DESKTOP-IJPEUEO:~$ ./answer
Memory from 0 to 15 Allocate
Memory from 32 to 63 Allocated
Memory from 64 to 95 Allocate
Memory block from 0 to 15 freed
Joining of the blocks starting at 0 and 16 was done by Varun
Sorry, the command cant be executed due to invalid request
Memory block from 32 to 63 freed
Joining of blocks starting at 0 and 32 was done by Varun
Sorry, the command cant be executed due to invalid request
varun@DESKTOP-IJPEUEO:~$
```

Question 3

a particular sized buffer serves as the queue for both the producer and consumer. The producer has the responsibility to generate data and store it in the buffer and at the same time, not keep on generating data once the buffer is full. On the consumer side, the job is to consume the buffer's data one by one and at the same time keep in mind that it doesn't keep on consuming even after the buffer goes empty.

Pseudocode Solution using Semaphore and Mutex

Initialization

Mutex mutex; // Used to provide mutual exclusion for critical section

Semaphore empty = N; // Number of empty slots in buffer

Semaphore full = 0 // Number of slots filled

int in = 0; //index at which producer will put the next data

int out = 0; // index from which the consumer will consume next data

int buffer[N];

Producer Code

```
while(True) {
  // produce an item
  wait(empty); // wait/sleep when there are no empty slots
```

```
  wait(mutex);
  buffer[in] = item
  in = (in+1)%buffersize;
  signal(mutex);
  signal(full); // Signal/wake to consumer that buffer has some  data and they can consume now
}
```

Consumer Code

```
while(True) {
  wait(full); // wait/sleep when there are no full slots
  wait(mutex);
  item = buffer[out];
  out = (out+1)%buffersize;
  signal(mutex);
  signal(empty); // Signal/wake the producer that buffer slots are emptied and they can produce more
  //consumer the item
}
```

```
varun@DESKTOP-IJPEUEO:~$ gcc -pthread PRODUCER-CONSUMER.c
varun@DESKTOP-IJPEUEO:~$ ./a.out
Producer 1: Insert Item 1804289383 at 0
Producer 1: Insert Item 1681692777 at 1
Producer 3: Insert Item 1714636915 at 2
Producer 2: Insert Item 846930886 at 3
Consumer 2: Remove Item 1804289383 from 0
Consumer 3: Remove Item 1681692777 from 1
Consumer 1: Remove Item 1714636915 from 2
Consumer 4: Remove Item 846930886 from 3
Producer 1: Insert Item 1957747793 at 0
Consumer 2: Remove Item 1957747793 from 0
Producer 3: Insert Item 719885386 at 1
Producer 1: Insert Item 596516649 at 2
Producer 2: Insert Item 1649760492 at 3
Consumer 3: Remove Item 719885386 from 1
Consumer 3: Remove Item 596516649 from 2
Producer 2: Insert Item 1025202362 at 0
Producer 4: Insert Item 424238335 at 1
Producer 3: Insert Item 1189641421 at 2
Consumer 1: Remove Item 1649760492 from 3
Consumer 2: Remove Item 1025202362 from 0
Consumer 3: Remove Item 424238335 from 1
Producer 3: Insert Item 1102520059 at 3
Producer 2: Insert Item 1350490027 at 0
Consumer 4: Remove Item 1189641421 from 2
Consumer 4: Remove Item 1102520059 from 3
Producer 4: Insert Item 783368690 at 1
Consumer 1: Remove Item 1350490027 from 0
Consumer 2: Remove Item 783368690 from 1
Producer 4: Insert Item 2044897763 at 2
Producer 4: Insert Item 1967513926 at 3
Consumer 4: Remove Item 2044897763 from 2
Consumer 1: Remove Item 1967513926 from 3
varun@DESKTOP-IJPEUEO:~$
```