Operating Systems

LABORATORY FILE

Submitted by:

Kshitisha Negi

SAP ID: 500107178

Batch: 9

B.Tech Computer Science and Engineering

Specialization - AIML

3rd Semester

Submitted to:

Abhijit Kumar

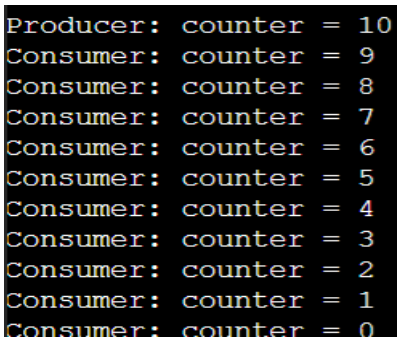
Experiment no 4: Semaphore

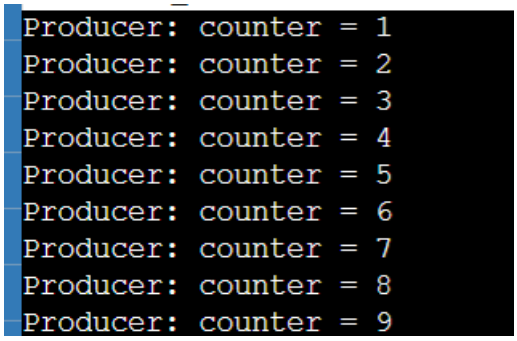
i)Write a program that demonstrates how two processes can share a variable using semaphore.

```c
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <pthread.h>
int counter = 0;
sem_t sem;
void *producer(void *arg)
{
  int i;

  for (i = 0; i < 10; i++) {
    sem_wait(&sem);
    counter++;
    printf("Producer: counter = %d\n", counter);
    sem_post(&sem);
  }

  pthread_exit(NULL);
}

void *consumer(void *arg)
{
  int i;

  for (i = 0; i < 10; i++) {
    sem_wait(&sem);
    counter--;
    printf("Consumer: counter = %d\n", counter);
    sem_post(&sem);
  }

  pthread_exit(NULL);
}

int main()
{
  pthread_t producer_thread, consumer_thread;
```

```c
  sem_init(&sem, 0, 1);

  pthread_create(&producer_thread, NULL, producer, NULL);
  pthread_create(&consumer_thread, NULL, consumer, NULL);

  pthread_join(producer_thread, NULL);
  pthread_join(consumer_thread, NULL);

  sem_destroy(&sem);

  return 0;
}
```

Output:





ii)To write a C program to implement the Producer & consumer Problem (Semaphore)

```c
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <pthread.h>

#define BUFFER_SIZE 10

typedef struct {
  int buffer[BUFFER_SIZE];
  int in;
  int out;
  sem_t empty;
  sem_t full;
} buffer_t;

buffer_t buffer;

void *producer(void *arg) {
  int item;

  for (int i = 0; i < 10; i++) {
    item = rand() % 100;

    sem_wait(&buffer.empty);

    buffer.buffer[buffer.in] = item;
    buffer.in = (buffer.in + 1) % BUFFER_SIZE;

    printf("Producer: produced %d\n", item);

    sem_post(&buffer.full);
  }

  pthread_exit(NULL);
}
```

```c
void *consumer(void *arg) {
  int item;

  for (int i = 0; i < 10; i++) {
    sem_wait(&buffer.full);

    item = buffer.buffer[buffer.out];
    buffer.out = (buffer.out + 1) % BUFFER_SIZE;

    printf("Consumer: consumed %d\n", item);

    sem_post(&buffer.empty);
  }

  pthread_exit(NULL);
}

int main() {
  pthread_t producer_thread, consumer_thread;

  sem_init(&buffer.empty, 0, BUFFER_SIZE);
  sem_init(&buffer.full, 0, 0);

  pthread_create(&producer_thread, NULL, producer, NULL);
  pthread_create(&consumer_thread, NULL, consumer, NULL);

  pthread_join(producer_thread, NULL);
  pthread_join(consumer_thread, NULL);

  sem_destroy(&buffer.empty);
  sem_destroy(&buffer.full);

  return 0;
}
```

Output:

```
Producer: produced 83
Producer: produced 86
Producer: produced 77
Producer: produced 15
Producer: produced 93
Producer: produced 35
Producer: produced 86
Producer: produced 92
Producer: produced 49
Producer: produced 21
Consumer: consumed 83
```

```
Consumer: consumed 86
Consumer: consumed 77
Consumer: consumed 15
Consumer: consumed 93
Consumer: consumed 35
Consumer: consumed 86
Consumer: consumed 92
Consumer: consumed 49
Consumer: consumed 21
```

Experiment no 5: Memory management-1

To write a C program to implement memory management using paging technique

```c
#include <stdio.h>

#define PAGE_SIZE 10
#define MEMORY_SIZE 100
#define NUMBER_OF_PAGES 10
#define NUMBER_OF_FRAMES 5
typedef struct {
  int page_number;
  int frame_number;
} page_table_entry;
page_table_entry page_table[NUMBER_OF_PAGES];
int memory[MEMORY_SIZE];
int free_frames[NUMBER_OF_FRAMES];
void initialize() {
  int i;
for (i = 0; i < NUMBER_OF_PAGES; i++) {
    page_table[i].page_number = -1;
    page_table[i].frame_number = -1;
  }
for (i = 0; i < MEMORY_SIZE; i++) {
    memory[i] = -1;
  }
for (i = 0; i < NUMBER_OF_FRAMES; i++) {
    free_frames[i] = i;
  }
}int translate_address(int logical_address) {
  int page_number = logical_address / PAGE_SIZE;
  int offset = logical_address % PAGE_SIZE;
if (page_table[page_number].frame_number == -1) {
    // Page fault
```

```c
if (free_frames[0] == -1) {
    // No free frames available
    return -1;
  }

  // Allocate a free frame
  int frame_number = free_frames[0];
  free_frames[0] = free_frames[1];
// Update page table
  page_table[page_number].page_number = page_number;
  page_table[page_number].frame_number = frame_number;
 }int physical_address = page_table[page_number].frame_number * PAGE_SIZE + offset;
 return physical_address;
}
int main() {
 int logical_address;
initialize();
printf("Enter logical address: ");
scanf("%d", &logical_address);
int physical_address = translate_address(logical_address);
if (physical_address == -1) {
  printf("Page fault occurred.\n");
 } else {
  printf("Physical address: %d\n", physical_address);
 }return 0;
}
```

Output:

```
PS C:\Users\ruhi\Desktop\Programming\OS_C> gcc exp5_1.c
PS C:\Users\ruhi\Desktop\Programming\OS_C> ./a.exe
Enter logical address: 43
Physical address: 3
```

Experiment no 7: FILE MANIPULATION

i) Displays the file and Directory

```
kshitisha@kshitisha-virtual-machine:~$ nano exp7_1.c
kshitisha@kshitisha-virtual-machine:~$ gcc exp7_1.c
```

```
  GNU nano 6.2
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int main(){
        char dir[20];
        int c = 0;

        c=strcpy(dir,"ls -l");
        system(dir);

        if(c==0)
        {
                printf("command didnt execute");
        }
        else
        {
                printf("command executed ");
        }
return 0;
}
```

```
kshitisha@kshitisha-virtual-machine:~$ ./a.out
total 72
-rwxrwxr-x 1 kshitisha kshitisha 16056 Dec  5 22:44 a.out
drwxr-xr-x 3 kshitisha kshitisha  4096 Dec  5 22:32 Desktop
drwxrwxr-x 2 kshitisha kshitisha  4096 Dec  1 15:28 desktop2
drwxr-xr-x 2 kshitisha kshitisha  4096 Sep 12 22:43 Documents
drwxr-xr-x 2 kshitisha kshitisha  4096 Oct  6 16:10 Downloads
-rw-rw-r-- 1 kshitisha kshitisha     0 Dec  5 22:01 exp7_1
-rw-rw-r-- 1 kshitisha kshitisha   340 Dec  5 22:44 exp7_1.c
drwxrwxr-x 2 kshitisha kshitisha  4096 Dec  5 22:42 exp7_kshitisha
drwxr-xr-x 2 kshitisha kshitisha  4096 Sep 12 22:43 Music
drwxrwxr-x 4 kshitisha kshitisha  4096 Sep 14 20:25 mydirectory
drwxr-xr-x 3 kshitisha kshitisha  4096 Sep 12 23:21 Pictures
drwxr-xr-x 2 kshitisha kshitisha  4096 Sep 11 19:40 Public
drwxrwxr-x 2 kshitisha kshitisha  4096 Sep 29 15:21 sjf
drwx------ 5 kshitisha kshitisha  4096 Sep 12 21:26 snap
drwxr-xr-x 2 kshitisha kshitisha  4096 Sep 11 19:40 Templates
drwxr-xr-x 2 kshitisha kshitisha  4096 Sep 11 19:40 Videos
```

ii)Creating new Directory

```
  GNU nano 6.2
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>

int main()
{
        char dirname[50];
        int c =0;

        printf("Enter the name of directory-");
        scanf("%s",&dirname);
        c = mkdir(dirname,777);

if (c == 0) {
        printf("Directory %s created successfully.\n", dirname);
    } else {
        printf("Error creating directory");
    }

    return 0;
}
```

```
                char *
kshitisha@kshitisha-virtual-machine:~$ ./a.out
Enter the name of directory- Kshitishaaa
Directory Kshitishaaa created successfully.
kshitisha@kshitisha-virtual-machine:~$
```

Experiment no 9: Deadlock avoidance

To implement Banker's algorithm for a multiple resources

```c
#include <stdio.h>
#include <stdlib.h>

#define NUM_PROCESSES 5 // Change this to the actual number of processes
#define NUM_RESOURCES 3 // Change this to the actual number of resources
int main() {
  // Initialize data structures
  int available[NUM_RESOURCES]; // Available resources
  int max[NUM_PROCESSES][NUM_RESOURCES]; // Maximum demand of each process
  int allocated[NUM_PROCESSES][NUM_RESOURCES]; // Resources allocated to each process
  int need[NUM_PROCESSES][NUM_RESOURCES]; // Need of each process (max - allocated)
  int finish[NUM_PROCESSES]; // Whether a process has finished or not
// Input available resources
  printf("Enter available resources: ");
  for (int i = 0; i < NUM_RESOURCES; i++) {
    scanf("%d", &available[i]);
  }
// Input maximum demand of each process
  printf("\nEnter maximum demand of each process:\n");
  for (int i = 0; i < NUM_PROCESSES; i++) {
    printf("Process %d: ", i);
    for (int j = 0; j < NUM_RESOURCES; j++) {
      scanf("%d", &max[i][j]);
    }
  }// Input resources allocated to each process
  printf("\nEnter resources allocated to each process:\n");
  for (int i = 0; i < NUM_PROCESSES; i++) {
    printf("Process %d: ", i);
    for (int j = 0; j < NUM_RESOURCES; j++) {
      scanf("%d", &allocated[i][j]);
    }
  }// Calculate need of each process
  for (int i = 0; i < NUM_PROCESSES; i++) {
    for (int j = 0; j < NUM_RESOURCES; j++) {
      need[i][j] = max[i][j] - allocated[i][j];
    }
  }
```

```
  for (int i = 0; i < NUM_PROCESSES; i++) {
    finish[i] = 0;
  }// Check for safe state
  int safe = 1;
  int work[NUM_RESOURCES]; // Temporary work vector
  for (int i = 0; i < NUM_RESOURCES; i++) {
    work[i] = available[i];
  } while (1) {
    int found = 0;
    for (int i = 0; i < NUM_PROCESSES; i++) {
      if (!finish[i]) {
        int need_satisfied = 1;
        for (int j = 0; j < NUM_RESOURCES; j++) {
          if (need[i][j] > work[j]) {
            need_satisfied = 0;
            break;
          }}
if (need_satisfied) {
          finish[i] = 1;
          for (int j = 0; j < NUM_RESOURCES; j++) {
            work[j] += allocated[i][j];
          }
          found = 1;
          break;
        }
      }
    }

    if (!found) {
      safe = 0;
      break;
    }
  }
```

```
  if (safe) {
    printf("\nSystem is in safe state.\n");
  } else {
    printf("\nSystem is not in safe state.\n");
  }

  return 0;
}
```

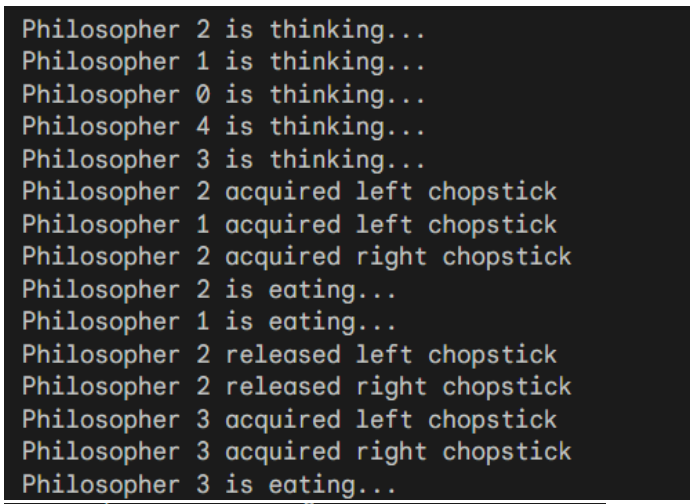
Output:

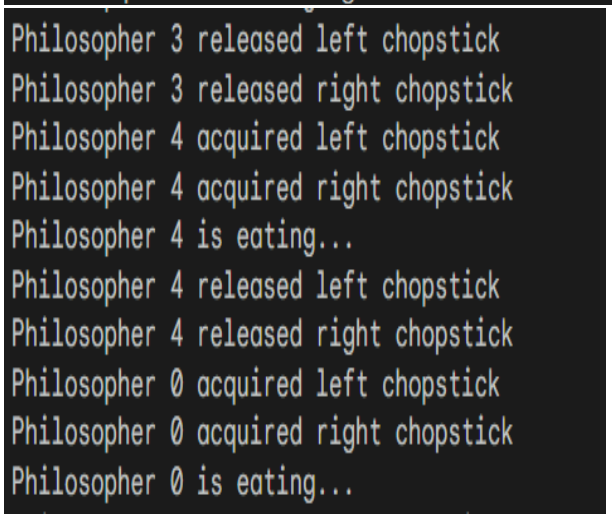
```
Enter available resources: 5
^[[A

Enter maximum demand of each process:
Process 0: Process 1: Process 2: Process 3: Process 4:
Enter resources allocated to each process:
Process 0: Process 1: Process 2: Process 3: Process 4:
System is not in safe state.
```

ii) To implement dinning philosopher's problem.

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define NUM_PHILOSOPHERS 5

sem_t chopsticks[NUM_PHILOSOPHERS];

void think() {
  printf("Philosopher %d is thinking...\n", (int)pthread_self());
  sleep(rand() % 5);
}

void eat() {
  printf("Philosopher %d is eating...\n", (int)pthread_self());
  sleep(rand() % 5);
}

void* philosopher(void* arg) {
  int philosopher_id = (int)arg;

  while (1) {
    think();

    // Acquire the left chopstick
    sem_wait(&chopsticks[(philosopher_id + 1) % NUM_PHILOSOPHERS]);

    printf("Philosopher %d acquired left chopstick\n", philosopher_id);

    // Check if the right chopstick is available
    if (sem_trywait(&chopsticks[philosopher_id])) {
      // Right chopstick is available, eat now
      eat();

      // Release both chopsticks
      sem_post(&chopsticks[philosopher_id]);
      sem_post(&chopsticks[(philosopher_id + 1) % NUM_PHILOSOPHERS]);
    } else {
      // Right chopstick is not available, release the left chopstick and try again later
      printf("Philosopher %d released left chopstick\n", philosopher_id);
      sem_post(&chopsticks[(philosopher_id + 1) % NUM_PHILOSOPHERS]);
    }
  }

  return NULL;
}

int main() {
  pthread_t philosophers[NUM_PHILOSOPHERS];

  // Initialize semaphores
  for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
    sem_init(&chopsticks[i], 0, 1);
  }

  // Create philosopher threads
  for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
    pthread_create(&philosophers[i], NULL, philosopher, (void*)i);
  }

  // Wait for all philosopher threads to finish
  for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
    pthread_join(philosophers[i], NULL);
  }

  return 0;
}
```

Output:


Philosopher 2 is thinking...
Philosopher 1 is thinking...
Philosopher 0 is thinking...
Philosopher 4 is thinking...
Philosopher 3 is thinking...
Philosopher 2 acquired left chopstick
Philosopher 1 acquired left chopstick
Philosopher 2 acquired right chopstick
Philosopher 2 is eating...
Philosopher 1 is eating...
Philosopher 2 released left chopstick
Philosopher 2 released right chopstick
Philosopher 3 acquired left chopstick
Philosopher 3 acquired right chopstick
Philosopher 3 is eating...


Philosopher 3 released left chopstick
Philosopher 3 released right chopstick
Philosopher 4 acquired left chopstick
Philosopher 4 acquired right chopstick
Philosopher 4 is eating...
Philosopher 4 released left chopstick
Philosopher 4 released right chopstick
Philosopher 0 acquired left chopstick
Philosopher 0 acquired right chopstick
Philosopher 0 is eating...


Philosopher 0 released left chopstick
Philosopher 0 released right chopstick
Philosopher 3 acquired left chopstick
Philosopher 3 acquired right chopstick
Philosopher 3 is eating...
Philosopher 3 released left chopstick
Philosopher 3 released right chopstick