```
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <pthread.h>

#define num_phil 5

typedef enum { thinking, hungry, eating } State;

typedef struct{
   State state[num_phil];
   pthread_cond_t self[num_phil];
   pthread_mutex_t mutex;
} DiningPhilosophers;

DiningPhilosophers dp;

void test(int i){
   if(dp.state[(i+4)%num_phil] != eating && dp.state[(i+1)%num_phil] !=eating && dp.state[i]==hungry){
      dp.state[i]=eating;
      pthread_cond_signal(&dp.self[i]);
   }
}

void pickup(int i){
   pthread_mutex_lock(&dp.mutex);
   dp.state[i] = hungry;
   test(i);
   while(dp.state[i]!=eating){
      pthread_cond_wait(&dp.self,&dp.mutex);
   }
   pthread_mutex_unlock(&dp.mutex);
}
```

```
void putdown(int i){
   pthread_mutex_lock(&dp.mutex);
   dp.state[i] = thinking;
   test((i+4)%num_phil);
   test((i+1)%num_phil);
   pthread_mutex_unlock(&dp.mutex);
}

void* philosopher(void* num){
   int i = *((int*)num);
   while(true){
      printf("Philosopher %d is thinking.\n",i);
      sleep(1);

      pickup(i);

      printf("Philosopher %d is eating\n",i);
      sleep(1);

      putdown(i);
   }
   return NULL;
}



int main()
{
   pthread_t philosophers[num_phil];
   dp.mutex = (pthread_mutex_t)PTHREAD_MUTEX_INITIALIZER;

   for(int i = 0; i < num_phil; i++){
      dp.state[i] = thinking;
      pthread_cond_init(&dp.self[i],NULL);
   }
```

```
    int philosopher_numbers[num_phil];

    for(int i = 0; i < num_phil; i++){
        philosopher_numbers[i] = i;
        pthread_create(&philosophers[i],NULL,philosopher,&philosopher_numbers[i]);
    }

    for(int i = 0; i < num_phil; i++){
        pthread_join(philosophers[i], NULL);
    }




    return 0;
}
```

Producer consumer

```
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <pthread.h>


#define buffer_size 5

int buffer[buffer_size];
int in = 0;
int out = 0;
int counter = 0;

pthread_mutex_t mutex;
pthread_cond_t not_empty;
pthread_cond_t not_full;

void* producer(void* arg){
   int next_produced = 0;
   while(true){
      while(counter==buffer_size){
         /* do nothing */
      }

      pthread_mutex_lock(&mutex);
      buffer[in] = next_produced;
      in = (in+1)%buffer_size;
      counter++;
      printf("produced %d\n", next_produced);
      pthread_cond_signal(&not_empty);
      pthread_mutex_unlock(&mutex);
```

```
        next_produced++;
        sleep(1);
    }
    return NULL;
}

void* consumer(void* arg){
    int next_consumed = 0;
    while(true){
        while(counter==0){
            /*do nothing*/
        }
        pthread_mutex_lock(&mutex);
        int next_consumed = buffer[out];
        out = (out+1)%buffer_size;
        counter--;
        printf("consumed %d\n",next_consumed);
        pthread_cond_signal(&not_full);
        pthread_mutex_unlock(&mutex);

        next_consumed++;
        sleep(1);

    }
    return NULL;
}


int main()
{
    pthread_t prod_thread, cons_thread;
    pthread_mutex_init(&mutex, NULL);
    pthread_mutex_init(&not_full,NULL);
    pthread_mutex_init(&not_empty,NULL);
```

```
    pthread_create(&prod_thread,NULL,producer,NULL);
    pthread_create(&cons_thread,NULL,consumer,NULL);

    pthread_join(prod_thread,NULL);
    pthread_join(cons_thread,NULL);

    return 0;
}
```

Page replacement

```
#include <bits/stdc++.h>

using namespace std;

// FIFO
int FIFO(vector<int>& pages, int numFrames) {
    unordered_set<int> pageInMemory;
    queue<int> pageQueue;
    int pageFaults = 0;

    for (int page : pages) {
        //if the page is not in memory, page fault
        if (pageInMemory.find(page) == pageInMemory.end()) {
            pageFaults++;

            //if memory is full, replace the oldest page
            if (pageQueue.size() == numFrames) {
                int oldestPage = pageQueue.front();
                pageQueue.pop();
                pageInMemory.erase(oldestPage);
```

```
            }

            pageQueue.push(page);
            pageInMemory.insert(page);
        }
    }

    return pageFaults;
}

// LRU
int LRU(vector<int>& pages, int numFrames) {
    unordered_map<int, list<int>::iterator> pageInMemory;
    list<int> recentPages;
    int pageFaults = 0;

    for (int page : pages) {
        //if the page is not in memory, page fault
        if (pageInMemory.find(page) == pageInMemory.end()) {
            pageFaults++;

            //if memory is full, replace the least recently used page
            if (recentPages.size() == numFrames) {
                int lruPage = recentPages.back();
                recentPages.pop_back();
                pageInMemory.erase(lruPage);
            }

            recentPages.push_front(page);
            pageInMemory[page] = recentPages.begin();
        } else {
            //if the page is in memory, move it to the front (most recently used)
            recentPages.erase(pageInMemory[page]);
            recentPages.push_front(page);
```

```
            pageInMemory[page] = recentPages.begin();
        }
    }

    return pageFaults;
}

// Optimal
int Optimal(vector<int>& pages, int numFrames) {
    unordered_set<int> pageInMemory;
    int pageFaults = 0;

    for (int i = 0; i < pages.size(); i++) {
        int page = pages[i];

        //if the page is not in memory, page fault
        if (pageInMemory.find(page) == pageInMemory.end()) {
            pageFaults++;

            //if memory is full, replace a page
            if (pageInMemory.size() == numFrames) {
                int farthestPage = -1, farthestIndex = -1;

                for (auto& entry : pageInMemory) {
                    int j;
                    for (j = i + 1; j < pages.size(); j++) {
                        if (pages[j] == entry) {
                            break;
                        }
                    }

                    // If the page is not going to be used again
                    if (j == pages.size()) {
                        farthestPage = entry;
```

```
                break;
            }

            //else track the farthest use
            if (j > farthestIndex) {
                farthestIndex = j;
                farthestPage = entry;
            }
        }

        //remove the page that will be used the farthest or not at all
        pageInMemory.erase(farthestPage);
      }

      pageInMemory.insert(page);
    }
  }

  return pageFaults;
}

int main() {
  vector<int> pages = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 4};
  int numFrames = 3;

  cout << "Page reference string: ";
  for (int page : pages) {
    cout << page << " ";
  }
  cout << endl;

  // FIFO
  int fifoFaults = FIFO(pages, numFrames);
  cout << "FIFO Page Faults: " << fifoFaults << endl;
```

```
    // LRU
    int lruFaults = LRU(pages, numFrames);
    cout << "LRU Page Faults: " << lruFaults << endl;

    // Optimal
    int optimalFaults = Optimal(pages, numFrames);
    cout << "Optimal Page Faults: " << optimalFaults << endl;

    return 0;
}
```

Memory allocation

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class MemoryAllocator {
public:
    MemoryAllocator(vector<int>& memory) : memory(memory) {}

    // First-Fit Allocation
    bool firstFit(int processSize) {
        for (int i = 0; i < memory.size(); ++i) {
            if (memory[i] >= processSize) {
                memory[i] -= processSize;
                cout << "First-fit: Allocated " << processSize << " to block " << i << " (Remaining space: " << memory[i]
<< ")\n";
                return true;
            }
        }
        cout << "First-fit: No suitable block found for size " << processSize << "\n";
```

```
    return false;
}

// Best-Fit Allocation
bool bestFit(int processSize) {
    int bestIndex = -1;
    int bestSize = INT_MAX;

    for (int i = 0; i < memory.size(); ++i) {
        if (memory[i] >= processSize && memory[i] - processSize < bestSize) {
            bestSize = memory[i] - processSize;
            bestIndex = i;
        }
    }

    if (bestIndex != -1) {
        memory[bestIndex] -= processSize;
        cout << "Best-fit: Allocated " << processSize << " to block " << bestIndex << " (Remaining space: " << memory[bestIndex] << ")\n";
        return true;
    }

    cout << "Best-fit: No suitable block found for size " << processSize << "\n";
    return false;
}

// Worst-Fit Allocation
bool worstFit(int processSize) {
    int worstIndex = -1;
    int worstSize = -1;

    for (int i = 0; i < memory.size(); ++i) {
        if (memory[i] >= processSize && memory[i] > worstSize) {
            worstSize = memory[i];
            worstIndex = i;
```

```
            }
        }

        if (worstIndex != -1) {
            memory[worstIndex] -= processSize;
            cout << "Worst-fit: Allocated " << processSize << " to block " << worstIndex << " (Remaining space: " << memory[worstIndex] << ")\n";
            return true;
        }

        cout << "Worst-fit: No suitable block found for size " << processSize << "\n";
        return false;
    }

    void displayMemoryState() {
        cout << "Current memory state: ";
        for (int i = 0; i < memory.size(); ++i) {
            cout << "[" << memory[i] << "] ";
        }
        cout << endl;
    }

private:
    vector<int> memory;
};

int main() {

    vector<int> memory = {100, 500, 200, 300, 600, 400};
    MemoryAllocator allocator(memory);

    cout << "Initial memory state:\n";
    allocator.displayMemoryState();

    vector<int> processSizes = {212, 417, 112, 426};
```

```
    // First-Fit Allocation
    cout << "\n---- First-Fit Allocation ----\n";
    for (int size : processSizes) {
        allocator.firstFit(size);
    }
    allocator.displayMemoryState();

    // Best-Fit Allocation
    cout << "\n---- Best-Fit Allocation ----\n";
    for (int size : processSizes) {
        allocator.bestFit(size);
    }
    allocator.displayMemoryState();

    // Worst-Fit Allocation
    cout << "\n---- Worst-Fit Allocation ----\n";
    for (int size : processSizes) {
        allocator.worstFit(size);
    }
    allocator.displayMemoryState();

    return 0;
}
```

1. Consider 2 processes. One process writes two values on to the shared memory and the other process reads the values and performs the sum of the values written.

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <semaphore.h>
#include <string.h>
#include <sys/wait.h>

#define SHM_NAME "/my_shared_memory"
#define SEM_WRITER "/sem_writer"
#define SEM_READER "/sem_reader"

typedef struct {
    int values[2];
} SharedData;

void writer(sem_t *sem_reader) {
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    ftruncate(shm_fd, sizeof(SharedData));
    SharedData *data = mmap(0, sizeof(SharedData), PROT_WRITE, MAP_SHARED, shm_fd, 0);

    //writing values to shared memory
    data->values[0] = 10;
    data->values[1] = 20;
    printf("Writer wrote values: %d, %d\n", data->values[0], data->values[1]);

    //signal reader
    sem_post(sem_reader);
    munmap(data, sizeof(SharedData));
    close(shm_fd);
}

void reader(sem_t *sem_reader) {
    int shm_fd = shm_open(SHM_NAME, O_RDONLY, 0666);
    SharedData *data = mmap(0, sizeof(SharedData), PROT_READ, MAP_SHARED, shm_fd, 0);

    //waiting for writer to write
    sem_wait(sem_reader);
    printf("Reader read values: %d, %d\n", data->values[0], data->values[1]);
    printf("Sum: %d\n", data->values[0] + data->values[1]);

    munmap(data, sizeof(SharedData));
    close(shm_fd);
}

int main() {
    //semaphores
    sem_t *sem_reader = sem_open(SEM_READER, O_CREAT, 0666, 0);
    if (sem_reader == SEM_FAILED) {
        perror("Semaphore creation failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        //child process: reader
        reader(sem_reader);
    } else {
        //parent process: writer
        writer(sem_reader);
        wait(NULL);  //waiting for child process to finish
    }

    //clean-up
    sem_close(sem_reader);
    sem_unlink(SEM_READER);
    shm_unlink(SHM_NAME);

    return 0;
}
```

```
Input
Writer wrote values: 10, 20
Reader read values: 10, 20
Sum: 30


...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment – 2

AIM: To understand and practice shell scripting

Shell Scripts

• Given the value of “n” in the command line, find the Fibonacci series

```
read n

a=0
b=1

for (( i=0; i<n; i++ )); do
    echo -n "$a "
    fn=$((a + b))
    a=$b
    b=$fn
done

echo
```

```
Input
5
0 1 1 2 3


...Program finished with exit code 0
Press ENTER to exit console.
```

• Read the value of “n” with prompt in the script and find the factorial of “n”

main.bash

```bash
read -p "Enter a number to find its factorial: " n

factorial=1

for (( i=1; i<=n; i++ )); do
    factorial=$((factorial * i))
done

echo "The factorial of $n is $factorial."
```

input

```
Enter a number to find its factorial: 5
The factorial of 5 is 120.


...Program finished with exit code 0
Press ENTER to exit console.
```

Find the average of the given numbers read in the command line - #.of .values and the numbers

Run | Debug | Stop | Share | Save | { } Beautify | Language Bash

main.bash

```bash
read -p "Enter the number of values: " count

sum=0

for (( i=1; i<=count; i++ )); do
    read -p "Enter number $i: " number
    sum=$((sum + number))
done

average=$((sum / count))

echo "Total Numbers: $count"
echo "Sum: $sum"
echo "Average: $average"
```

input

```
Enter the number of values: 5
Enter number 1: 3
Enter number 2: 5
Enter number 3: 6
Enter number 4: 7
Enter number 5: 1
Total Numbers: 5
Sum: 22
Average: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

• Create a Menu driven calculator using case….esac for performing +,-,* and /

main.bash

```bash
while true; do

    echo "Menu:"
    echo "1. Addition (+)"
    echo "2. Subtraction (-)"
    echo "3. Multiplication (*)"
    echo "4. Division (/)"
    echo "5. Exit"

    read -p "Select an option (1-5): " choice

    case $choice in
        1)
            read -p "Enter first number: " num1
            read -p "Enter second number: " num2
            result=$((num1 + num2))
            echo "Result: $num1 + $num2 = $result"
            ;;
        2)
            read -p "Enter first number: " num1
            read -p "Enter second number: " num2
            result=$((num1 - num2))
            echo "Result: $num1 - $num2 = $result"
            ;;
        3)
            read -p "Enter first number: " num1
            read -p "Enter second number: " num2
            result=$((num1 * num2))
            echo "Result: $num1 * $num2 = $result"
            ;;
```

main.bash

```bash
            echo "Result: $num1 - $num2 = $result"
            ;;
        3)
            read -p "Enter first number: " num1
            read -p "Enter second number: " num2
            result=$((num1 * num2))
            echo "Result: $num1 * $num2 = $result"
            ;;
        4)
            read -p "Enter first number: " num1
            read -p "Enter second number: " num2
            if [ $num2 -eq 0 ]; then
                echo "Error: Division by zero is not allowed."
            else
                result=$((num1 / num2))
                echo "Result: $num1 / $num2 = $result"
            fi
            ;;
        5)
            echo "Exiting the calculator."
            exit 0
            ;;
        *)
            echo "Invalid option. Please select a number between 1 and 5."
            ;;
    esac

    echo ""
done
```

input

```
Menu:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Select an option (1-5): 1
Enter first number: 3
Enter second number: 4
Result: 3 + 4 = 7

Menu:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Select an option (1-5): 2
Enter first number: 2
Enter second number: 3
Result: 2 - 3 = -1
```

```
Menu:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Select an option (1-5): 3
Enter first number: 2
Enter second number: 3
Result: 2 * 3 = 6

Menu:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Select an option (1-5): 44
Invalid option. Please select a number between 1 and 5.

Menu:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Select an option (1-5): 4
Enter first number: 4
Enter second number: 2
Result: 4 / 2 = 2
```

```
Menu:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Exit
Select an option (1-5): 5
Exiting the calculator.


...Program finished with exit code 0
Press ENTER to exit console.
```

- Read a number on the command line and check if it is an Armstrong number or not

main.bash

```bash
read -p "Enter a number " number

sum=0
temp=$number
num_digits=${#number}

while [ $temp -gt 0 ]; do
    digit=$((temp % 10))
    sum=$((sum + digit ** num_digits))
    temp=$((temp / 10))
done

if [ $sum -eq $number ]; then
    echo "$number is an Armstrong number."
else
    echo "$number is not an Armstrong number."
fi
```

input

```
Enter a number 153
153 is an Armstrong number.


...Program finished with exit code 0
Press ENTER to exit console.
```

- Print the multiplication table of the number given