System call

#include <iostream>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <cstdlib>

using namespace std;

// Bubble Sort in Ascending Order

void bubblesortasc(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] > arr[j+1]) {

swap(arr[j], arr[j+1]);

}

}

}

}

// Bubble Sort in Descending Order

void bubblesortdsc(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] < arr[j+1]) {

swap(arr[j], arr[j+1]);

}

}

}

}

int main() {

int n = 6;

int arr[n] = {1, 3, 2, 4, 5, 6};

// Display initial array

cout << "Initial Array: ";

for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

cout << endl;

pid\_t pid = fork();

if (pid < 0) {

// Error in forking

perror("fork failed");

exit(1);

} else if (pid == 0) {

// Child process

cout << "Child process" << endl;

bubblesortasc(arr, n); // Sort in ascending order

// Print sorted array in ascending order

cout << "Array after sorting (ascending order): ";

for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

cout << endl;

cout << "PID of child: " << getpid() << endl;

printf("Child process exiting...\n");

sleep(5); // Simulating delay before exit

\_exit(0); // Exit the child process

} else {

// Parent process

cout << "Parent process" << endl;

bubblesortdsc(arr, n); // Sort in descending order

// Print sorted array in descending order

cout << "Array after sorting (descending order): ";

for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

cout << endl;

cout << "PID of parent: " << getpid() << endl;

printf("Parent process exiting...\n");

wait(NULL); // Wait for child process to finish

sleep(5); // Simulating delay before exit

}

return 0;

}

Optimal page replacement

#include <stdio.h>

int findOptimal(int pages[], int temp[], int n, int index) {

    int farthest = index, pos = -1, i, j;

    for (i = 0; i < n; ++i) {

        int found = 0;

        for (j = index; j < n; ++j) {

            if (pages[j] == temp[i]) {

                if (j > farthest) {

                    farthest = j;

                    pos = i;

                }

                found = 1;

                break;

            }

        }

        if (!found)

            return i;  // If a page is not found in the future references

    }

    return (pos == -1) ? 0 : pos;  // If all pages are found in future references

}

int main() {

    int n, frames, pages[30], temp[10], pageFaults = 0;

    printf("Enter number of pages: ");

    scanf("%d", &n);

    printf("Enter reference string: ");

    for (int i = 0; i < n; ++i) {

        scanf("%d", &pages[i]);

    }

    printf("Enter number of frames: ");

    scanf("%d", &frames);

    for (int i = 0; i < frames; ++i) {

        temp[i] = -1;

    }

    for (int i = 0; i < n; ++i) {

        int pageHit = 0;

        for (int j = 0; j < frames; ++j) {

            if (temp[j] == pages[i]) {

                pageHit = 1;

                break;

            }

        }

        if (!pageHit) {

            if (i < frames) {

                temp[i] = pages[i];

            } else {

                int pos = findOptimal(pages, temp, frames, i);

                temp[pos] = pages[i];

            }

            pageFaults++;

        }

        printf("\n");

        for (int j = 0; j < frames; ++j) {

            printf("%d\t", temp[j]);

        }

    }

    printf("\n\nTotal Page Faults = %d\n", pageFaults);

    return 0;

}

2.FCFS Page replacement

#include <iostream>

using namespace std;

int main()

{

    int pages[30],n,frames,temp[10],pagefound,emptyslot,pos=0,fault=0;

    cout <<"Enter no. of pages\n";

    cin>>n;

    cout<<"Enter reference string \n";

    for(int i=0;i<n;i++){

        cin>>pages[i];

    }

    cout<<"Enter no. of frames\n";

    cin>>frames;

    for(int i=0;i<frames;i++){

        temp[i]=-1;

    }

    for(int i=0;i<n;i++){

        pagefound=0;

        for(int j=0;j<frames;j++){

            if(temp[j]==pages[i]){

                pagefound=1;

                break;

            }

        }

        if(pagefound==0){

            emptyslot=0;

            for(int j=0;j<frames;j++){

                if(temp[j]==-1){

                    fault++;

                    temp[j]=pages[i];

                    emptyslot=1;

                    break;

                }

            }

        if(emptyslot==0){

            temp[pos]=pages[i];

            pos++;

            if(pos>=frames){

                pos=0;

            }

            fault++;

        }

    }

        cout<<"\n";

        for(int j=0;j<frames;j++){

            cout<<temp[j]<<" ";

        }

    }

        cout<<"\n";

        cout<<"Page faults are :"<<fault;

    return 0;

}

3.LRU

#include <stdio.h>

int findLRU(int time[], int n) {

int i, minimum = time[0], pos = 0;

for (i = 1; i < n; ++i) {

if (time[i] < minimum) {

minimum = time[i];

pos = i;

}

}

return pos;

}

int main() {

int n, frames, pages[30], temp[10], counter = 0, time[10], pageHit, emptyFrame, i, j, pos, pageFaults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter reference string: ");

for (i = 0; i < n; ++i) {

scanf("%d", &pages[i]);

}

printf("Enter number of frames: ");

scanf("%d", &frames);

for (i = 0; i < frames; ++i) {

temp[i] = -1;

}

for (i = 0; i < n; ++i) {

pageHit = emptyFrame = 0;

for (j = 0; j < frames; ++j) {

if (temp[j] == pages[i]) {

counter++;

time[j] = counter;

pageHit = 1;

break;

}

}

if (pageHit == 0) {

for (j = 0; j < frames; ++j) {

if (temp[j] == -1) {

counter++;

pageFaults++;

temp[j] = pages[i];

time[j] = counter;

emptyFrame = 1;

break;

}

}

}

if (emptyFrame == 0) {

pos = findLRU(time, frames);

counter++;

pageFaults++;

temp[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for (j = 0; j < frames; ++j) {

printf("%d\t", temp[j]);

}

}

printf("\n\nTotal Page Faults = %d\n", pageFaults);

return 0;

}

4.Address Book

# Online Bash Shell.

# Code, Compile, Run and Debug Bash script online.

# Write your code in this editor and press "Run" button to execute it.

echo "Hello World";

add\_records(){

read -p "ID :" ID

read -p "Name : " Name

read -p "Phone :" Phone

echo "$ID" "$Name" "$Phone" >> address.txt

echo "Entry created!"

}

search\_records(){

record=$1

grep -w "$record" address.txt | while read -r id name phone; do

echo "ID - $id name-$name phone-$phone"

done

}

delete\_records(){

read -p "Enter record to be deleted - " data

sed -i "/$data/d" address.txt

echo "Deleted Successfully."

}

modify(){

read -p "wht to modify- " data

read -p "Enter new ID: " NEW\_ID

read -p "Enter new Name: " NEW\_NAME

read -p "Enter new Phone: " NEW\_PHONE

sed -i "/$data/c\\$NEW\_ID $NEW\_NAME $NEW\_PHONE" address.txt

}

read choice

case "$choice" in

1)add\_records ;;

2)read -p "what to search :" num ;search\_records "$num";;

3)delete\_records;;

4)modify;;

Esac

5.Bankers Algo

#include <iostream>

using namespace std;

int main(){

    int n=5;

    int r=3;

    int alloc[5][3]={{0,0,1},{3,0,0},{1,0,0},{2,3,2},{0,0,3}};

    int max[5][3]={{7,6,3},{3,2,2},{8,0,2},{2,1,2},{5,2,3}};

    int avail[3]={2,3,2};

    int need[n][r];

    for(int i=0;i<n;i++){

        for(int j=0;j<r;j++){

            need[i][j]=max[i][j]-alloc[i][j];

        }

    }

    int f[n],ans[n],ind=0;

    for(int k=0;k<n;k++){

        f[k]=0;

    }

    for(int k=0;k<5;k++){

        for(int i=0;i<n;i++){

            if(f[i]==0){

            int flag=0;

            for(int j=0;j<r;j++){

                if(need[i][j]>avail[j]){

                    flag=1;

                    break;

                }

            }

            if(flag==0){

                ans[ind++]=i;

                for(int y=0;y<r;y++){

                    avail[y]+=alloc[i][y];

                }

                f[i]=1;

            }

        }

    }

}

cout<<"The safe sequence are : "<<endl;

for(int i=0;i<n;i++){

    cout<<"P"<<ans[i]<<"->";

}

return 0;

}

6.SJF

#include <iostream>

#include <climits>

using namespace std;

void calculateTimes(int n, int at[], int bt[], int wt[], int tat[]) {

    int rem\_bt[n], complete = 0, t = 0;

    for (int i = 0; i < n; i++) rem\_bt[i] = bt[i];

    while (complete != n) {

        int shortest = -1, minm = INT\_MAX;

        for (int i = 0; i < n; i++) {

            if (at[i] <= t && rem\_bt[i] > 0 && rem\_bt[i] < minm) {

                minm = rem\_bt[i];

                shortest = i;

            }

        }

        // If no process found, increment time and continue

        if (shortest == -1) { t++; continue; }

        // Process the shortest job for one time unit

        rem\_bt[shortest]--;

        t++;

        // If process is finished, calculate waiting time

        if (rem\_bt[shortest] == 0) {

            complete++;

            wt[shortest] = t - bt[shortest] - at[shortest];

            if (wt[shortest] < 0) wt[shortest] = 0;  // Ensure no negative waiting time

        }

    }

    // Calculate turnaround times

    for (int i = 0; i < n; i++) tat[i] = bt[i] + wt[i];

}

int main() {

    int n = 4;

    int arrival\_time[] = {0, 1, 2, 4};

    int burst\_time[] = {5, 4, 2, 1};

    int wt[n] = {}, tat[n] = {};

    calculateTimes(n, arrival\_time, burst\_time, wt, tat);

    // Display results and calculate averages

    float total\_wt = 0, total\_tat = 0;

    cout << "Process  Arrival  Burst  Waiting  Turnaround\n";

    for (int i = 0; i < n; i++) {

        total\_wt += wt[i];

        total\_tat += tat[i];

        cout << " " << i + 1 << "\t    " << arrival\_time[i] << "\t    " << burst\_time[i]

             << "\t    " << wt[i] << "\t    " << tat[i] << endl;

    }

    cout << "\nAvg waiting time = " << total\_wt / n;

    cout << "\nAvg turnaround time = " << total\_tat / n;

    return 0;

}

7.RR

#include <iostream>

#include <algorithm>

using namespace std;

void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[], int timeQuantum) {

int rem\_bt[n];

for (int i = 0; i < n; i++) {

rem\_bt[i] = bt[i]; // Initialize remaining burst times

}

int t = 0; // Current time

bool done;

while (true) {

done = true;

for (int i = 0; i < n; i++) {

if (rem\_bt[i] > 0 && at[i] <= t) { // Process has arrived and has remaining burst time

done = false; // There is a pending process

if (rem\_bt[i] > timeQuantum) {

t += timeQuantum;

rem\_bt[i] -= timeQuantum;

} else {

t += rem\_bt[i];

wt[i] = t - bt[i] - at[i];

rem\_bt[i] = 0;

}

}

}

// If no process was processed in this cycle, increment time

if (done) {

for (int i = 0; i < n; i++) {

if (rem\_bt[i] > 0 && at[i] > t) {

t = at[i]; // Move to next process arrival time

done = false;

break;

}

}

}

if (done) break; // All processes are complete

}

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

for (int i = 0; i < n; i++) {

tat[i] = bt[i] + wt[i];

}

}

int main() {

int n = 3;

int processes[] = {1, 2, 3};

int burst\_time[] = {10, 5, 8};

int arrival\_time[] = {0, 1, 2};

int timeQuantum = 2;

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, burst\_time, wt, arrival\_time, timeQuantum);

findTurnAroundTime(processes, n, burst\_time, wt, tat);

cout << "Processes Arrival Time Burst Time Waiting Time Turn Around Time\n";

for (int i = 0; i < n; i++) {

total\_wt += wt[i];

total\_tat += tat[i];

cout << " " << processes[i] << "\t\t" << arrival\_time[i] << "\t\t" << burst\_time[i]

<< "\t\t" << wt[i] << "\t\t" << tat[i] << endl;

}

cout << "\nAverage waiting time = " << (float)total\_wt / (float)n;

cout << "\nAverage turn around time = " << (float)total\_tat / (float)n;

return 0;

}

SCAN –

#include <stdio.h>

#include <stdlib.h>

int main() {

    int n, i, j, head, size, seek = 0, diff;

    int queue[20], temp, max, min, index;

    printf("Enter the size of the disk: ");

    scanf("%d", &size);

    printf("Enter the number of requests: ");

    scanf("%d", &n);

    printf("Enter the initial head position: ");

    scanf("%d", &head);

    printf("Enter the request sequence: ");

    for (i = 0; i < n; i++) {

        scanf("%d", &queue[i]);

    }

    // Sort the request array

    for (i = 0; i < n; i++) {

        for (j = i + 1; j < n; j++) {

            if (queue[i] > queue[j]) {

                temp = queue[i];

                queue[i] = queue[j];

                queue[j] = temp;

            }

        }

    }

    // Find the index of the first request greater than the head

    for (i = 0; i < n; i++) {

        if (queue[i] > head) {

            index = i;

            break;

        }

    }

    // Service the requests to the right of the head

    for (i = index; i < n; i++) {

        diff = abs(queue[i] - head);

        seek += diff;

        head = queue[i];

    }

    // Move the head to the end of the disk

    seek += abs(size - queue[i - 1] - 1);

    head = size - 1;

    // Service the requests to the left of the head

    for (i = index - 1; i >= 0; i--) {

        diff = abs(queue[i] - head);

        seek += diff;

        head = queue[i];

    }

    printf("Total seek time: %d\n", seek);

    return 0;

}

Sstf

#include <stdio.h>

#include <stdlib.h>

int main() {

    int n, i, j, head, size, seek = 0, diff;

    int queue[20], serviced[20], index;

    printf("Enter the size of the disk: ");

    scanf("%d", &size);

    printf("Enter the number of requests: ");

    scanf("%d", &n);

    printf("Enter the initial head position: ");

    scanf("%d", &head);

    printf("Enter the request sequence: ");

    for (i = 0; i < n; i++) {

        scanf("%d", &queue[i]);

        serviced[i] = 0; // Mark all requests as not serviced

    }

    // Loop through all requests

    for (i = 0; i < n; i++) {

        int min = 10000;

        // Find the request with the minimum seek time from the current head

        for (j = 0; j < n; j++) {

            if (!serviced[j]) {

                diff = abs(queue[j] - head);

                if (diff < min) {

                    min = diff;

                    index = j;

                }

            }

        }

        // Service the request

        seek += min;

        head = queue[index];

        serviced[index] = 1; // Mark this request as serviced

    }

    printf("Total seek time: %d\n", seek);

    return 0;

}

C look

#include <stdio.h>

#include <stdlib.h>

int main() {

    int n, i, j, head, size, seek = 0, diff;

    int queue[20], temp, index;

    printf("Enter the size of the disk: ");

    scanf("%d", &size);

    printf("Enter the number of requests: ");

    scanf("%d", &n);

    printf("Enter the initial head position: ");

    scanf("%d", &head);

    printf("Enter the request sequence: ");

    for (i = 0; i < n; i++) {

        scanf("%d", &queue[i]);

    }

    // Sort the request array

    for (i = 0; i < n; i++) {

        for (j = i + 1; j < n; j++) {

            if (queue[i] > queue[j]) {

                temp = queue[i];

                queue[i] = queue[j];

                queue[j] = temp;

            }

        }

    }

    // Find the index of the first request greater than the head

    for (i = 0; i < n; i++) {

        if (queue[i] > head) {

            index = i;

            break;

        }

    }

    // Service the requests to the right of the head

    for (i = index; i < n; i++) {

        diff = abs(queue[i] - head);

        seek += diff;

        head = queue[i];

    }

    // Jump to the first request on the left side without adding to seek time

    if (index > 0) {

        seek += abs(queue[0] - head);

        head = queue[0];

    }

    // Service the requests to the left of the initial head position

    for (i = 1; i < index; i++) {

        diff = abs(queue[i] - head);

        seek += diff;

        head = queue[i];

    }

    printf("Total seek time: %d\n", seek);

    return 0;

}

System call

#include <iostream>

#include <cstdlib>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#define MAX\_SIZE 10

// Function to compare two integers (used for sorting)

int compare(const void \*a, const void \*b) {

return (\*(int \*)a - \*(int \*)b);

}

int main() {

int arr[MAX\_SIZE];

pid\_t pid;

char \*args[MAX\_SIZE + 2]; // +2 for program name and NULL

// Accept the array from the user

std::cout << "Enter " << MAX\_SIZE << " integers:\n";

for (int i = 0; i < MAX\_SIZE; i++) {

std::cin >> arr[i];

}

// Sort the array using qsort

qsort(arr, MAX\_SIZE, sizeof(int), compare);

// Create a child process using fork()

pid = fork();

if (pid < 0) {

// Error in forking

perror("Fork failed");

exit(1);

}

if (pid > 0) {

// Parent process

// Wait for child process to complete

wait(NULL);

} else {

// Child process

// Convert the sorted array to string arguments for execve

for (int i = 0; i < MAX\_SIZE; i++) {

args[i] = new char[20]; // Allocate memory for each argument

snprintf(args[i], 20, "%d", arr[i]); // Convert integer to string

}

args[MAX\_SIZE] = nullptr; // Terminate the argument list

// Execute the new program that will display the array in reverse order

execve("./reverse\_display", args, nullptr);

// If execve fails

perror("execve failed");

exit(1);

}

return 0;

}

Reverse

#include <iostream>

#include <cstdlib>

#define MAX\_SIZE 10

int main(int argc, char \*argv[]) {

if (argc <= 1) {

std::cout << "No array passed!\n";

return 1;

}

std::cout << "Array in Reverse Order: ";

// Display the array in reverse order

for (int i = argc - 1; i >= 1; i--) {

std::cout << argv[i] << " ";

}

std::cout << std::endl;

return 0;

}

8.using mutex

#include <pthread.h>

#include <stdio.h>

pthread\_mutex\_t wrt; // Mutex for writer's access

pthread\_mutex\_t mutex; // Mutex for controlling access to the numreader count

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

pthread\_mutex\_lock(&wrt); // Lock the writer mutex

cnt = cnt \* 2;

printf("Writer %d modified cnt to %d\n", \*((int \*)wno), cnt);

pthread\_mutex\_unlock(&wrt); // Unlock the writer mutex

}

void \*reader(void \*rno)

{

pthread\_mutex\_lock(&mutex); // Lock the mutex to modify numreader

numreader++;

if (numreader == 1) {

pthread\_mutex\_lock(&wrt); // First reader locks the writer

}

pthread\_mutex\_unlock(&mutex); // Unlock the mutex

// Reading Section

printf("Reader %d: read cnt as %d\n", \*((int \*)rno), cnt);

pthread\_mutex\_lock(&mutex); // Lock the mutex to modify numreader

numreader--;

if (numreader == 0) {

pthread\_mutex\_unlock(&wrt); // Last reader releases the writer

}

pthread\_mutex\_unlock(&mutex); // Unlock the mutex

}

int main()

{

pthread\_t read[10], write[5];

pthread\_mutex\_init(&wrt, NULL); // Initialize mutex for writer access

pthread\_mutex\_init(&mutex, NULL); // Initialize mutex for reader count

int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for (int i = 0; i < 10; i++) {

pthread\_create(&read[i], NULL, reader, (void \*)&a[i]);

}

for (int i = 0; i < 5; i++) {

pthread\_create(&write[i], NULL, writer, (void \*)&a[i]);

}

for (int i = 0; i < 10; i++) {

pthread\_join(read[i], NULL);

}

for (int i = 0; i < 5; i++) {

pthread\_join(write[i], NULL);

}

pthread\_mutex\_destroy(&wrt);

pthread\_mutex\_destroy(&mutex);

return 0;

}

9.using semaphore

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

sem\_t wrt; // Semaphore for writer's access

sem\_t mutex; // Semaphore for controlling access to the numreader count

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt); // Wait for access to modify cnt

cnt = cnt \* 2;

printf("Writer %d modified cnt to %d\n", \*((int \*)wno), cnt);

sem\_post(&wrt); // Release access to writer

}

void \*reader(void \*rno)

{

sem\_wait(&mutex); // Lock the mutex to modify numreader

numreader++;

if (numreader == 1) {

sem\_wait(&wrt); // First reader locks the writer

}

sem\_post(&mutex); // Unlock the mutex

// Reading Section

printf("Reader %d: read cnt as %d\n", \*((int \*)rno), cnt);

sem\_wait(&mutex); // Lock the mutex to modify numreader

numreader--;

if (numreader == 0) {

sem\_post(&wrt); // Last reader releases the writer

}

sem\_post(&mutex); // Unlock the mutex

}

int main()

{

pthread\_t read[10], write[5];

sem\_init(&wrt, 0, 1); // Initialize semaphore for writer (binary semaphore)

sem\_init(&mutex, 0, 1); // Initialize semaphore for reader count mutex

int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for (int i = 0; i < 10; i++) {

pthread\_create(&read[i], NULL, reader, (void \*)&a[i]);

}

for (int i = 0; i < 5; i++) {

pthread\_create(&write[i], NULL, writer, (void \*)&a[i]);

}

for (int i = 0; i < 10; i++) {

pthread\_join(read[i], NULL);

}

for (int i = 0; i < 5; i++) {

pthread\_join(write[i], NULL);

}

sem\_destroy(&wrt);

sem\_destroy(&mutex);

return 0;

}