Arch Linux - Virtual x86

copy.sh/v86/

Pause | Exit | Send Ctrl-Alt-Del | Send Alt-Tab | Insert floppy image | Save State | Load State | Memory Dump | Capture network traffic | Disable mouse | Lock mouse | Go fullscreen | Take screenshot | Mute | Scale: 1.0

```
root@localhost:~# ls
docker.sh  hello.c   hello.ml  hello.py  nbench         startx.sh  v86-in-v86.js
hello.asm  hello.js  hello.pl  hello.rs  networking.sh  v86
root@localhost:~# wc Mehul
wc: Mehul: No such file or directory
root@localhost:~# wc hello.c
 7 13 89 hello.c
root@localhost:~# cal 4 2003
     April 2003
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

root@localhost:~# who
root     tty1         Oct  5 19:08
root     ttyS0        Jan 13 22:41
root@localhost:~# date
Mon Nov  7 03:01:19 UTC 2022
root@localhost:~# pwd
/root
root@localhost:~#
```

Running: 9m 59s
Speed: 0.5 mIPS
Avg speed: 0.6 mIPS

9p Filesystem
Bytes read: 635186
Bytes written: 0
Last file: date
Status: Idle

VGA
Mode: Graphical
Resolution: 1024x768
BPP: 32

Mouse: Yes

Send files to emulator
Choose Files No file chosen

Get file from emulator
Absolute path

This is the serial console. Whatever you type or paste here will be sent to COM1

Arch Linux - Virtual x86

copy.sh/v86/

Pause | Exit | Send Ctrl-Alt-Del | Send Alt-Tab | Insert floppy image | Save State | Load State | Memory Dump | Capture network traffic | Disable mouse | Lock mouse | Go fullscreen | Take screenshot | Mute | Scale: 1.0

```
#include <stdio.h>

main()
{
    printf("Hello, world\n");
}
root@localhost:~# cp hello.c mehul_copy.c
root@localhost:~# cat mehul_copy.c
// to run: tcc -run hello.c
#include <stdio.h>

main()
{
    printf("Hello, world\n");
}
root@localhost:~# rm mehul_copy.c
root@localhost:~# cat mehul_copy.c
cat: mehul_copy.c: No such file or directory
root@localhost:~# mv hello.c mehul_move
root@localhost:~# cat hello.c
cat: hello.c: No such file or directory
root@localhost:~# cat mehul_move
// to run: tcc -run hello.c
#include <stdio.h>

main()
{
    printf("Hello, world\n");
}
root@localhost:~# cmp hllo.c mehul_copy.c
cmp: hllo.c: No such file or directory
root@localhost:~# cmp hello.c mehul_copy.c
cmp: hello.c: No such file or directory
root@localhost:~# find mehul_move
mehul_move
root@localhost:~# grep "e" mehul_move
// to run: tcc -run hello.c
#include <stdio.h>
    printf("Hello, world\n");
root@localhost:~# awk '{print $1}' mehul_move
//
#include

main()
{
printf("Hello,
}
root@localhost:~#
```

```
Running: 17m 57s
Speed: 0.5 mIPS
Avg speed: 0.7 mIPS

9p Filesystem
Bytes read: 3742862
Bytes written: 89
Last file: libm.so.6
Status: Idle

VGA
Mode: Graphical
Resolution: 1024x768
BPP: 32

Mouse: Yes
```

Send files to emulator

Choose Files No file chosen

Get file from emulator

Absolute path

```
This is the serial console. Whatever you type or paste here will be sent to COM1
```

Arch Linux - Virtual x86

copy.sh/v86/

Pause | Exit | Send Ctrl-Alt-Del | Send Alt-Tab | Insert floppy image | Save State | Load State | Memory Dump | Capture network traffic | Disable mouse | Lock mouse | Go fullscreen | Take screenshot | Mute | Scale: 1.0

```
root@localhost:~# mkdir mehul
root@localhost:~# cd mhul
-bash: cd: mhul: No such file or directory
root@localhost:~# cd mehul
root@localhost:~/mehul# mkdir new
root@localhost:~/mehul# ls
new
root@localhost:~/mehul# rmdir new
root@localhost:~/mehul# ls
root@localhost:~/mehul# _
```

```
Running: 24m 00s
Speed: 0.7 mIPS
Avg speed: 0.6 mIPS

9p Filesystem
Bytes read: 3844295
Bytes written: 89
Last file: mehul
Status: Idle

VGA
Mode: Graphical
Resolution: 1024x768
BPP: 32

Mouse: Yes
```

Send files to emulator

Choose Files No file chosen

Get file from emulator

Absolute path

This is the serial console. Whatever you type or paste here will be sent to COM1

Practical 2:

fork():

The Fork system call is used for creating a new process Unix systems, which is called the(child process), which runs concurrently with the process that makes the fork() call (parent process).

Different values returned by fork()**:**

- **Negative Value**: The creation of a child process was unsuccessful.
- **Zero**: Returned to the newly created child process.
- **Positive value**: Returned to parent or caller.

Open

process.c
~/Desktop

```c
#include<sys/wait.h>
#include<stdio.h>
#include<unistd.h>

using namespace std;

int main(){
    pid_t a;
    a= fork();
    printf("Process ID (PID=%d)\n",getpid());

    if(a<0){
        fprintf(stderr, "Fork Failed",getpid());
        return 1;
    }

    else if(a==0){
        execlp("/bin/ls","ls",NULL,getpid());
    }

    else{
        wait(NULL);
        printf("Child created",getpid());
        system("ps");
    }
    return 0;
}
```

After running the code in terminal:

```
root@localhost-VirtualBox:~/Desktop$ ./a.out
Process ID (PID=8971)
Process ID (PID=8972)
a.out  copy.cpp  dest_file.txt  fork.cpp  process.c  src_file.txt
    PID TTY          TIME CMD
   8909 pts/0    00:00:00 bash
   8971 pts/0    00:00:00 a.out
   8973 pts/0    00:00:00 sh
   8974 pts/0    00:00:00 ps
Child created root@localhost-VirtualBox:~/Desktop$
```

Kill():

Built-in command which is used to terminate processes manually.

Syntax:

kill [signal] PID

Parameters:

- PID =The `kill` command requires the process ID (PID) of the process we want to terminate.
- [signal] =Specify the signal.

Text Editor Sep 19 19:11

Open kill.c ~/Desktop

```c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<signal.h>

int main(int argc, int *argv[]){
    printf("The pid is = %d\n",getpid());
    sleep(5);
    kill(getpid(), SIGSEGV);
return 0;
}
```

After running the code in terminal:

```
root@localhost-VirtualBox:~/Desktop$ gcc kill.c
root@localhost-VirtualBox:~/Desktop$ ./a.out
The pid is = 10225
Segmentation fault (core dumped)
root@localhost-VirtualBox:~/Desktop$
```

Sleep():

Sets the process to wait until the specified amount of time proceeds. The sleep system call is used to take a time value as a parameter, specifying the minimum amount of time that the process is to sleep before resuming execution.

Syntax:

sleep(time);

getpid():

Returns the process ID of the calling process.

Syntax:

pid_t getpid(void);

Parameters:

Return type: getpid() returns the process ID of the current process. It never throws any error therefore is always successful.

```
root@localhost-VirtualBox:~/Desktop$ gcc kill.c
root@localhost-VirtualBox:~/Desktop$ ./a.out
The pid is = 10225
Segmentation fault (core dumped)
root@localhost-VirtualBox:~/Desktop$
```

Ps:

Viewing information related with the processes on a system which stands as abbreviation for “Process Status”.
ps command is used to list the currently running processes and their PIDs.

Syntax:

ps [options]

parameters:

- **PID –** The unique process ID
- **TTY –** Terminal type that the user is logged into
- **TIME –**Amount of CPU in minutes and seconds that the process has been running.
- **CMD –** Name of the command that launched the process.

```
  PID TTY          TIME CMD
 8909 pts/0    00:00:00 bash
 8971 pts/0    00:00:00 a.out
 8973 pts/0    00:00:00 sh
 8974 pts/0    00:00:00 ps
```

Practical 3:

```
//same program same code
#include <iostream>
#include <unistd.h>
using namespace std;

int main()
{
        pid_t pid;
        pid =fork();
        //cout<<pid<<endl;
        if (pid<0)
        {
                cout<<"error"<<endl;
        }
        else
        {
                cout<<"HELLO WORLD"<<endl;
        }
        return 0;
}
```

```
1 //same program different code
2 #include <iostream>
3 #include <unistd.h>
4 #include <sys/wait.h>
5
6 using namespace std;
7
8 int main()
9 {
10         int pid;
11         pid =fork();
12         //cout<<pid<<endl;
13         if (pid<0)
14         {
15                 cout<<"error"<<endl;
16         }
17         else if(pid==0)
18         {
19                 cout<<pid<<endl;
20                 cout<<"This is a child process"<<endl;
21         }
22         else if(pid>0)
23         {       wait(NULL);
24                 cout<<pid<<endl;
25                 cout<<"This is a parent process"<<endl;
26         }
27         return 0;
28 }
```

```
1 //same program different code
2 #include <iostream>
3 #include <unistd.h>
4 using namespace std;
5
6 int main()
7 {
8         int pid;
9         pid =fork();
10        //cout<<pid<<endl;
11        if (pid<0)
12        {
13                cout<<"error"<<endl;
14        }
15        else if(pid==0)
16        {
17                cout<<pid<<endl;
18                cout<<"This is a child process"<<endl;
19        }
20        else
21        {
22                cout<<pid<<endl;
23                cout<<"This is a parent process"<<endl;
24        }
25        return 0;
26 }
```

```
cssem3@cs-lab03-HP-ProDesk-600-G6-Microtower-PC:~/Desktop$ g++ forkos.cpp
cssem3@cs-lab03-HP-ProDesk-600-G6-Microtower-PC:~/Desktop$ ./a.out
HELLO WORLD
HELLO WORLD
cssem3@cs-lab03-HP-ProDesk-600-G6-Microtower-PC:~/Desktop$ g++ forkos2.cpp
cssem3@cs-lab03-HP-ProDesk-600-G6-Microtower-PC:~/Desktop$ ./a.out
11780
This is a parent process
0
This is a child process
cssem3@cs-lab03-HP-ProDesk-600-G6-Microtower-PC:~/Desktop$ g++ forkos3.cpp
cssem3@cs-lab03-HP-ProDesk-600-G6-Microtower-PC:~/Desktop$ ./a.out
0
This is a child process
11792
This is a parent process
```

Practical 4

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
using namespace std;


int main(){

system("cat /proc/cpuinfo");
}
```

```
root@localhost:/# g++ cpuinfo.cpp
root@localhost:/# ./a.out_
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 6
model name      : 0f/06
stepping        : 3
cpu MHz         : 1000.000
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fdiv_bug        : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu pse tsc msr pae cx8 sep pge cmov mmx fxsr sse sse2 constant_tsc tsc_reliable cpuid tsc_known_freq popcnt r
drand hypervisor erms
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit mmio_unknown
bogomips        : 2000.33
clflush size    : 32
cache_alignment : 32
address sizes   : 36 bits physical, 32 bits virtual
power management:
```

Practical 5

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
using namespace std;



int main(){

system("cat /proc/meminfo");
}
```

```
root@localhost:/# g++ meminfo.cpp
root@localhost:/# ./a.out
```

```
Inactive(file):     36140 kB
Unevictable:            0 kB
Mlocked:                0 kB
HighTotal:              0 kB
HighFree:               0 kB
LowTotal:          491200 kB
LowFree:           425956 kB
SwapTotal:              0 kB
SwapFree:               0 kB
Zswap:                  0 kB
Zswapped:               0 kB
Dirty:                  0 kB
Writeback:              0 kB
AnonPages:           2224 kB
Mapped:              6784 kB
Shmem:                 36 kB
KReclaimable:        6992 kB
Slab:               13780 kB
SReclaimable:        6992 kB
SUnreclaim:          6788 kB
KernelStack:          464 kB
PageTables:           120 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:       245600 kB
Committed_AS:        3952 kB
VmallocTotal:      487672 kB
VmallocUsed:         4780 kB
VmallocChunk:           0 kB
Percpu:               264 kB
HardwareCorrupted:      0 kB
AnonHugePages:          0 kB
ShmemHugePages:         0 kB
ShmemPmdMapped:         0 kB
FileHugePages:          0 kB
FilePmdMapped:          0 kB
CmaTotal:               0 kB
CmaFree:                0 kB
HugePages_Total:        0
HugePages_Free:         0
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        4096 kB
Hugetlb:                0 kB
DirectMap4k:        40704 kB
DirectMap4M:       483328 kB
```

Practical 6:

www.onlinegdb.com/online_c++_compiler

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp | source.txt | destination.txt

```cpp
#include <iostream>
#include <fcntl.h>
#include <unistd.h>

using namespace std;

bool copyFile(const char *sourceFile, const char *destinationFile) {
    int source_fd = open(sourceFile, O_RDONLY);
    if (source_fd == -1) {
        perror("Error opening source file");
        return false;
    }

    int dest_fd = open(destinationFile, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (dest_fd == -1) {
        perror("Error opening destination file");
        close(source_fd);
        return false;
    }

    char buffer[BUFSIZ];
    ssize_t bytesRead, bytesWritten;

    while ((bytesRead = read(source_fd, buffer, BUFSIZ)) > 0) {
        bytesWritten = write(dest_fd, buffer, (ssize_t)bytesRead);
        if (bytesWritten != bytesRead) {
            perror("Error writing to file");
            close(source_fd);
            close(dest_fd);
            return false;
        }
    }

    if (bytesRead == -1) {
        perror("Error reading from file");
        close(source_fd);
        close(dest_fd);
        return false;
    }

```

input

www.onlinegdb.com/online_c++_compiler

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

Language C++

main.cpp | source.txt | destination.txt

```cpp
    }

    if (bytesRead == -1) {
        perror("Error reading from file");
        close(source_fd);
        close(dest_fd);
        return false;
    }

    close(source_fd);
    close(dest_fd);
    return true;
}

int main() {
    const char *sourceFile = "source.txt";  // Replace with your source file path
    const char *destinationFile = "destination.txt";  // Replace with your destination file path

    if (copyFile(sourceFile, destinationFile)) {
        cout << "File copied successfully." << endl;
    } else {
        cerr << "Failed to copy file." << endl;
    }

    return 0;
}
```

input

```
File copied successfully.


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

www.onlinegdb.com/online_c++_compiler

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

Language C++

main.cpp | source.txt | destination.txt

```
hello everyone
Kaizoku Ou ni orewa naru!
```

input

```
File copied successfully.


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

https://www.onlinegdb.com/online_c++_compiler#_editor_6233763711

www.onlinegdb.com/online_c++_compiler

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

Language C++

main.cpp | source.txt | destination.txt

```
hello everyone
Kaizoku Ou ni orewa naru!
```

input

```
File copied successfully.


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

Practical 8: FCFS

```
#include<iostream>
using namespace std;

// A function to calculate waiting time of each processes
void cal_WT(int n, int wt[], int bt[]) {
    // waiting time of first job will always be 0
    wt[0]=0;
    for(int i=1; i<n; i++) {
        wt[i] = bt[i-1] + wt[i-1];
    }
}
// a function to calculate turn around time
void cal_TAT( int n, int wt[], int bt[], int TAT[]) {
    for(int i=0; i<n; i++) {
        TAT[i] = bt[i] + wt[i];
    }
}

// function to display data
void display(int p[], int n, int wt[], int bt[], int tat[]) {
    cout << "\nEvaluating processes using first come first serve (FCFS) algorithm: " << endl;
    cout << "\nProcess\t" << "Burst time\t" << "Waiting time\t" << "Turn Around time" << endl;

    float avg_wt=0;
    float avg_tat=0;
    for(int i=0; i<n; i++) {
        avg_wt += wt[i];
        avg_tat += tat[i];
        cout << "  " << p[i] << "       " << bt[i] << "\t\t" << wt[i] << "\t\t\t" << tat[i] << endl;
    }

    cout << "\n Average waiting time is: " << avg_wt/n << endl;
    cout << "\n Turn Around time is: " << avg_tat/n << endl;

}
int main(){
    int p_id[]  = {1, 2, 3, 4, 5};
    int BT [] = {5, 24, 16, 10, 3};
    int WT [5];
    int TAT[5];

    cal_WT(5,WT,BT);
    cal_TAT(5, WT, BT, TAT);

    display(p_id, 5, WT, BT, TAT);

return 0;
}
```

FCFS Output

www.onlinegdb.com/online_c++_compiler

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp

```cpp
#include<iostream>
using namespace std;

// A function to calculate waiting time of each processes
void cal_WT(int n, int wt[], int bt[]) {
    // waiting time of first job will always be 0
    wt[0]=0;
    for(int i=1; i<n; i++) {
        wt[i] = bt[i-1] + wt[i-1];
    }
}
// a function to calculate turn around time
void cal_TAT( int n, int wt[], int bt[], int TAT[]) {
    for(int i=0; i<n; i++) {
        TAT[i] = bt[i] + wt[i];
    }
}

// function to display data
```

input

```
Evaluating processes using first come first serve (FCFS) algorithm:

Process Burst time      Waiting time    Turn Around time
  1       5             0                       5
  2       24            5                       29
  3       16            29                      45
  4       10            45                      55
  5       3             55                      58

 Average waiting time is: 26.8

 Turn Around time is: 38.4


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

SJF: Code

www.onlinegdb.com/online_c++_compiler

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp

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

using namespace std;

struct Process {
    int id;
    int burst_time;
};

bool compare(Process a, Process b) {
    return a.burst_time < b.burst_time;
}

void SJF(vector<Process>& processes) {
    sort(processes.begin(), processes.end(), compare);

    int n = processes.size();
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    vector<int> waiting_time(n, 0);
    vector<int> turnaround_time(n, 0);

    waiting_time[0] = 0;
    turnaround_time[0] = processes[0].burst_time;

    for (int i = 1; i < n; i++) {
        waiting_time[i] = waiting_time[i - 1] + processes[i - 1].burst_time;
        turnaround_time[i] = waiting_time[i] + processes[i].burst_time;

        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }

    cout << "SJF Scheduling: " << endl;
    cout << "Process\tBurst Time\tWaiting Time\tTurnaround Time" << endl;

    for (int i = 0; i < n; i++) {
        cout << processes[i].id << "\t" << processes[i].burst_time << "\t\t"
             << waiting_time[i] << "\t\t" << turnaround_time[i] << endl;
    }

    cout << "\nAverage Waiting Time: " << (float)total_waiting_time / n << endl;
    cout << "Average Turnaround Time: " << (float)total_turnaround_time / n << endl;
}

int main() {
    vector<Process> processes = {{1, 6}, {2, 8}, {3, 7}, {4, 3}};

    SJF(processes);

    return 0;
```

input

```
Press ENTER to exit console.
```

Call Stack

#	Function	File:Line

Local Variables

Variable	Value

Registers

Register	Value

Display Expressions

Expression	Value

Enter expression to watch

Breakpoints and Watchpoints

#	Description

SJF Output

www.onlinegdb.com/online_c++_compiler

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp

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

using namespace std;

struct Process {
    int id;
    int burst_time;
```

input

```
SJF Scheduling:
Process Burst Time      Waiting Time    Turnaround Time
4       3               0               3
1       6               3               9
3       7               9               16
2       8               16              24

Average Waiting Time: 7
Average Turnaround Time: 12.25


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

Call Stack

#	Function	File:Line

Local Variables

Variable	Value

Registers

Register	Value

Display Expressions

Expression	Value

Enter expression to watch

Breakpoints and Watchpoints

#	Description

STRF code

www.onlinegdb.com/online_c++_compiler

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp

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

using namespace std;

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    bool completed;
};

bool compareArrival(Process a, Process b) {
    return a.arrival_time < b.arrival_time;
}

bool compareRemainingTime(Process a, Process b) {
    return a.remaining_time < b.remaining_time;
}

void SRTF(vector<Process>& processes) {
    sort(processes.begin(), processes.end(), compareArrival);

    int n = processes.size();
    int current_time = 0;
    int completed = 0;

    for (int i = 0; i < n; i++) {
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].completed = false;
    }

    while (completed != n) {
        int min_remaining = INT_MAX;
        int shortest_process = -1;

        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && !processes[i].completed &&
                processes[i].remaining_time < min_remaining) {
                min_remaining = processes[i].remaining_time;
                shortest_process = i;
            }
        }

        if (shortest_process == -1) {
```

input

```
Press ENTER to exit console.
```

STRF output

www.onlinegdb.com/online_c++_compiler

Run | Debug | Stop | Share | Save | { } Beautify | Language C++

main.cpp

```cpp
        }

        if (shortest_process == -1) {
            current_time++;
            continue;
        }

        processes[shortest_process].remaining_time--;
        min_remaining = processes[shortest_process].remaining_time;

        if (min_remaining == 0) {
            completed++;
            processes[shortest_process].completed = true;

            int turnaround_time = current_time - processes[shortest_process].arrival_time + 1;
            int waiting_time = turnaround_time - processes[shortest_process].burst_time;

            cout << "Process " << processes[shortest_process].id << " - Turnaround Time: "
                 << turnaround_time << ", Waiting Time: " << waiting_time << endl;
        }

        current_time++;
    }
}

int main() {
    vector<Process> processes = {{1, 0, 6, 0}, {2, 2, 8, 0}, {3, 4, 7, 0}, {4, 6, 3, 0}};

    SRTF(processes);

    return 0;
}
```

input

```
Process 1 - Turnaround Time: 6, Waiting Time: 0
Process 4 - Turnaround Time: 3, Waiting Time: 0
Process 3 - Turnaround Time: 12, Waiting Time: 5
Process 2 - Turnaround Time: 22, Waiting Time: 14


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

Practical 9

www.onlinegdb.com/online_c++_compiler#

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

Language C++

main.cpp

```cpp
#include <iostream>
#include <pthread.h>
#include <vector>

using namespace std;

struct ThreadData {
    vector<int> nums;
    int sum;
};

void *sumThread(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    data->sum = 0;

    for (int num : data->nums) {
        data->sum += num;
    }

    pthread_exit(NULL);
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;

    // Creating a list of n numbers (you can input or generate these numbers as needed)
    vector<int> numbers(n);
    cout << "Enter " << n << " numbers:\n";
    for (int i = 0; i < n; ++i) {
        cin >> numbers[i];
    }

    pthread_t threads[2];
    ThreadData data[2];

    // Dividing the list into two smaller lists of equal size
    int mid = n / 2;
    data[0].nums = vector<int>(numbers.begin(), numbers.begin() + mid);
```

input

```
Press ENTER to exit console.
```

www.onlinegdb.com/online_c++_compiler#

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

Language C++

main.cpp

```cpp
    cout << "Enter " << n << " numbers:\n";
    for (int i = 0; i < n; ++i) {
        cin >> numbers[i];
    }

    pthread_t threads[2];
    ThreadData data[2];

    // Dividing the list into two smaller lists of equal size
    int mid = n / 2;
    data[0].nums = vector<int>(numbers.begin(), numbers.begin() + mid);
    data[1].nums = vector<int>(numbers.begin() + mid, numbers.end());

    for (int i = 0; i < 2; ++i) {
        if (pthread_create(&threads[i], NULL, sumThread, (void *)&data[i]) != 0) {
            cerr << "Error creating thread" << endl;
            return 1;
        }
    }

    for (int i = 0; i < 2; ++i) {
        pthread_join(threads[i], NULL);
    }

    // Summing up the results from both threads
    int totalSum = data[0].sum + data[1].sum;
    cout << "Sum of the numbers: " << totalSum << endl;

    return 0;
}
```

input

```
Enter the number of elements: 5
Enter 5 numbers:
1 42 69 56 420
Sum of the numbers: 588


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

Practical 10:

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

class MemoryBlock {
public:
    int startAddress;
    int size;
    bool allocated;

    MemoryBlock(int start, int s) : startAddress(start), size(s), allocated(false) {}
};

class MemoryManager {
private:
    std::vector<MemoryBlock> memoryBlocks;

public:
    MemoryManager(std::vector<int> blockSizes) {
        int startAddress = 0;
        for (int size : blockSizes) {
            memoryBlocks.push_back(MemoryBlock(startAddress, size));
            startAddress += size;
        }
    }

    void displayMemory() {
        std::cout << "Memory Status:\n";
        for (const MemoryBlock& block : memoryBlocks) {
            std::cout << "Block at Address " << block.startAddress << ", Size " << block.size;
```

```cpp
        std::cout << "Memory Status:\n";
        for (const MemoryBlock& block : memoryBlocks) {
            std::cout << "Block at Address " << block.startAddress << ", Size " << block.size;
            if (block.allocated) {
                std::cout << " (Allocated)\n";
            } else {
                std::cout << " (Free)\n";
            }
        }
        std::cout << "\n";
    }

    void firstFit(int processSize) {
        for (MemoryBlock& block : memoryBlocks) {
            if (!block.allocated && block.size >= processSize) {
                block.allocated = true;
                std::cout << "First Fit: Allocated " << processSize << " at Address " << block.startAddress
                return;
            }
        }
        std::cout << "First Fit: Insufficient Memory\n";
    }

    void bestFit(int processSize) {
        auto bestFitBlock = std::min_element(memoryBlocks.begin(), memoryBlocks.end(),
            [processSize](const MemoryBlock& a, const MemoryBlock& b) {
                return a.size >= processSize && b.size >= processSize && a.size < b.size;
            });

        if (bestFitBlock != memoryBlocks.end() && bestFitBlock->size >= processSize) {
```

```
        if (bestFitBlock != memoryBlocks.end() && bestFitBlock->size >= processSize) {
            bestFitBlock->allocated = true;
            std::cout << "Best Fit: Allocated " << processSize << " at Address " << bestFitBlock->startAddres
        } else {
            std::cout << "Best Fit: Insufficient Memory\n";
        }
    }

    void worstFit(int processSize) {
        auto worstFitBlock = std::max_element(memoryBlocks.begin(), memoryBlocks.end(),
            [processSize](const MemoryBlock& a, const MemoryBlock& b) {
                return a.size >= processSize && b.size >= processSize && a.size < b.size;
            });

        if (worstFitBlock != memoryBlocks.end() && worstFitBlock->size >= processSize) {
            worstFitBlock->allocated = true;
            std::cout << "Worst Fit: Allocated " << processSize << " at Address " << worstFitBlock->startAddre
        } else {
            std::cout << "Worst Fit: Insufficient Memory\n";
        }
    }
};

int main() {
    // Initialize Memory Manager with block sizes
    MemoryManager memoryManager({100, 200, 50, 300, 150});

    // Display initial memory status
    memoryManager.displayMemory();
```

```
};

int main() {
    // Initialize Memory Manager with block sizes
    MemoryManager memoryManager({100, 200, 50, 300, 150});

    // Display initial memory status
    memoryManager.displayMemory();

    // Allocate memory using different strategies
    memoryManager.firstFit(120);
    memoryManager.displayMemory();

    memoryManager.bestFit(80);
    memoryManager.displayMemory();

    memoryManager.worstFit(200);
    memoryManager.displayMemory();

    return 0;
}
```

```
Memory Status:
Block at Address 0, Size 100 (Free)
Block at Address 100, Size 200 (Free)
Block at Address 300, Size 50 (Free)
Block at Address 350, Size 300 (Free)
Block at Address 650, Size 150 (Free)

First Fit: Allocated 120 at Address 100
Memory Status:
Block at Address 0, Size 100 (Free)
Block at Address 100, Size 200 (Allocated)
Block at Address 300, Size 50 (Free)
Block at Address 350, Size 300 (Free)
Block at Address 650, Size 150 (Free)

Best Fit: Allocated 80 at Address 0
Memory Status:
Block at Address 0, Size 100 (Allocated)
Block at Address 100, Size 200 (Allocated)
Block at Address 300, Size 50 (Free)
Block at Address 350, Size 300 (Free)
Block at Address 650, Size 150 (Free)

Worst Fit: Insufficient Memory
Memory Status:
Block at Address 0, Size 100 (Allocated)
Block at Address 100, Size 200 (Allocated)
Block at Address 300, Size 50 (Free)
Block at Address 350, Size 300 (Free)
Block at Address 650, Size 150 (Free)
```