31 **Worst Fit Memory Allocation**

#include <stdio.h>

void worstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

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

allocation[i] = -1;

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

int worstIdx = -1;

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

if (blockSize[j] >= processSize[i]) {

if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])

worstIdx = j;

}

}

if (worstIdx != -1) {

allocation[i] = worstIdx;

blockSize[worstIdx] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock No.\n");

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

printf("%d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main() {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = sizeof(blockSize) / sizeof(blockSize[0]);

int n = sizeof(processSize) / sizeof(processSize[0]);

worstFit(blockSize, m, processSize, n);

return 0;

}

OUTPUT

Process No. Process Size Block No.

1 212 5

2 417 2

3 112 5

4 426 Not Allocated

32 **Best Fit Memory Allocation**

#include <stdio.h>

void bestFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

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

allocation[i] = -1;

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

int bestIdx = -1;

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

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])

bestIdx = j;

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock No.\n");

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

printf("%d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main() {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = sizeof(blockSize) / sizeof(blockSize[0]);

int n = sizeof(processSize) / sizeof(processSize[0]);

bestFit(blockSize, m, processSize, n);

return 0;

}

OUTPUT

Process No. Process Size Block No.

1 212 3

2 417 2

3 112 1

4 426 5

33 **First Fit Memory Allocation**

#include <stdio.h>

void firstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

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

allocation[i] = -1;

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

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

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

printf("\nProcess No.\tProcess Size\tBlock No.\n");

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

printf("%d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main() {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = sizeof(blockSize) / sizeof(blockSize[0]);

int n = sizeof(processSize) / sizeof(processSize[0]);

firstFit(blockSize, m, processSize, n);

return 0;

}

OUTPUT

Process No. Process Size Block No.

1 212 2

2 417 5

3 112 2

4 426 Not Allocated

34 **UNIX System Calls for File Management**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd;

char buffer[100];

fd = open("testfile.txt", O\_CREAT | O\_RDWR, 0644);

if (fd < 0) {

perror("File open failed");

return 1;

}

write(fd, "Hello, UNIX File System!\n", 25);

lseek(fd, 0, SEEK\_SET);

read(fd, buffer, 25);

buffer[25] = '\0';

printf("Read from file: %s", buffer);

close(fd);

return 0;

}

OUTPUT

35 **UNIX I/O System Calls Implementation**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

int main() {

int fd = open("testfile.txt", O\_CREAT | O\_RDWR, 0644);

if (fd < 0) {

perror("Open failed");

return 1;

}

struct stat fileStat;

if (stat("testfile.txt", &fileStat) == 0)

printf("File Size: %ld bytes\n", fileStat.st\_size);

write(fd, "UNIX System Calls", 17);

lseek(fd, 0, SEEK\_SET);

char buffer[20];

read(fd, buffer, 17);

buffer[17] = '\0';

printf("File Content: %s\n", buffer);

close(fd);

DIR \*dir = opendir(".");

struct dirent \*entry;

printf("Current Directory Files:\n");

while ((entry = readdir(dir)) != NULL)

printf("%s\n", entry->d\_name);

closedir(dir);

return 0;

}

OUTPUT

File Size: 0 bytes

File Content: UNIX System Calls

Current Directory Files:

file1.txt

file2.c

testfile.txt

...

36 **File Management Operations**

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*fp = fopen("example.txt", "w");

if (fp == NULL) {

perror("File creation failed");

return 1;

}

fprintf(fp, "This is a sample file.");

fclose(fp);

fp = fopen("example.txt", "r");

char buffer[100];

fgets(buffer, 100, fp);

printf("File Content: %s\n", buffer);

fclose(fp);

if (remove("example.txt") == 0)

printf("File deleted successfully.\n");

else

printf("File deletion failed.\n");

return 0;

}

OUTPUT

File Content: This is a sample file.

File deleted successfully.

37 **Simulating the** ls **UNIX Command**

#include <stdio.h>

#include <dirent.h>

int main() {

DIR \*d;

struct dirent \*dir;

d = opendir(".");

if (d) {

printf("Files in current directory:\n");

while ((dir = readdir(d)) != NULL) {

printf("%s\n", dir->d\_name);

}

closedir(d);

} else {

perror("opendir failed");

}

return 0;

}

OUTPUT

Files in current directory:

file1.c

file2.txt

example.txt

...

38 **Simulating the** grep **UNIX Command**

#include <stdio.h>

#include <string.h>

int main() {

FILE \*fp = fopen("sample.txt", "r");

if (!fp) {

perror("File opening failed");

return 1;

}

char line[256], search[] = "keyword";

while (fgets(line, sizeof(line), fp)) {

if (strstr(line, search))

printf("%s", line);

}

fclose(fp);

return 0;

}

OUTPUT

(keyword found line from file)

39

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER\_SIZE 5

sem\_t empty, full;

pthread\_mutex\_t mutex;

int buffer[BUFFER\_SIZE], count = 0;

void \*producer(void \*arg) {

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

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[count++] = i;

printf("Produced: %d\n", i);

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

sleep(1);

}

return NULL;

}

void \*consumer(void \*arg) {

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

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

printf("Consumed: %d\n", buffer[--count]);

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

sleep(1);

}

return NULL;

}

int main() {

pthread\_t prod, cons;

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

pthread\_mutex\_init(&mutex, NULL);

pthread\_create(&prod, NULL, producer, NULL);

pthread\_create(&cons, NULL, consumer, NULL);

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

sem\_destroy(&empty);

sem\_destroy(&full);

pthread\_mutex\_destroy(&mutex);

return 0;

}

OUTPUT

Produced: 0

Consumed: 0

Produced: 1

Consumed: 1

Produced: 2

Consumed: 2

Produced: 3

Consumed: 3

Produced: 4

Consumed: 4

40 **Thread Operations: Create, Join, Equal, Exit**

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

void \*threadFunc(void \*arg) {

printf("Thread executing...\n");

pthread\_exit(NULL);

}

int main() {

pthread\_t thread1, thread2;

pthread\_create(&thread1, NULL, threadFunc, NULL);

pthread\_create(&thread2, NULL, threadFunc, NULL);

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

if (pthread\_equal(thread1, thread2))

printf("Threads are equal\n");

else

printf("Threads are not equal\n");

printf("Exiting main thread\n");

pthread\_exit(NULL);

}

OUTPUT

Thread executing...

Thread executing...

Threads are not equal

Exiting main thread