1. Create a Process and Display IDs

#include <stdio.h>

#include <unistd.h>

int main() {

if (fork() == 0) printf("Child: %d\n", getpid());

else printf("Parent: %d\n", getpid());

return 0;

}

2. File Copy Using System Calls

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

char src[100], dst[100], b[1024];

printf("Source: "); scanf("%s", src);

printf("Dest: "); scanf("%s", dst);

int s = open(src, O\_RDONLY), t = open(dst, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (s < 0 || t < 0) return perror("File error"), 1;

ssize\_t n;

while ((n = read(s, b, sizeof(b))) > 0) write(t, b, n);

close(s); close(t);

return 0;

}

3. FCFS CPU Scheduling

#include <stdio.h>

int main() {

int n = 3, b[] = {4, 3, 2}, w = 0, t = 0;

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

t += b[i]; printf("P%d W:%d T:%d\n", i+1, w, t); w += b[i];

}

return 0;

}

4. SJF Scheduling

#include <stdio.h>

int main() {

int n = 3, b[] = {4, 3, 2}, w = 0, t = 0;

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

if (b[j] > b[j+1]) { int tmp = b[j]; b[j] = b[j+1]; b[j+1] = tmp; }

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

t += b[i]; printf("P%d W:%d T:%d\n", i+1, w, t); w += b[i];

}

return 0;

}

5. Priority Scheduling

#include <stdio.h>

int main() {

int n = 3, b[] = {4, 3, 2}, p[] = {2, 1, 3}, w = 0, t = 0;

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

if (p[j] > p[j+1]) { int tmp = p[j]; p[j] = p[j+1]; b[j] ^= b[j+1] ^= b[j] ^= b[j+1]; }

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

t += b[i]; printf("P%d W:%d T:%d\n", i+1, w, t); w += b[i];

}

return 0;

}

6. Preemptive Priority Scheduling

#include <stdio.h>

int main() {

int n = 3, bt[] = {4, 3, 2}, at[] = {0, 1, 2}, p[] = {2, 1, 3}, wt = 0, t = 0;

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

printf("P%d: Start:%d End:%d\n", i+1, t, t+bt[i]);

t += bt[i];

}

return 0;

}

7. Non-Preemptive SJF Scheduling

#include <stdio.h>

int main() {

int n = 3, bt[] = {4, 3, 2}, wt = 0, t = 0;

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

if (bt[j] > bt[j+1]) { int tmp = bt[j]; bt[j] = bt[j+1]; bt[j+1] = tmp; }

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

printf("P%d: W:%d T:%d\n", i+1, wt, wt + bt[i]);

wt += bt[i];

}

return 0;

}

8. Round Robin Scheduling

#include <stdio.h>

int main() {

int n = 3, bt[] = {4, 3, 2}, tq = 2, rem[] = {4, 3, 2}, t = 0, done;

do {

done = 1;

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

if (rem[i] > 0) {

printf("P%d: Start:%d ", i+1, t);

int qt = rem[i] > tq ? tq : rem[i];

t += qt; rem[i] -= qt; done = 0;

printf("End:%d\n", t);

}

}

} while (!done);

return 0;

}

9. Inter-Process Communication Using Shared Memory

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

int main() {

int shmid = shmget(IPC\_PRIVATE, 1024, 0666|IPC\_CREAT);

char \*str = (char\*) shmat(shmid, NULL, 0);

strcpy(str, "Hello, Shared Memory!");

printf("Data Written: %s\n", str);

shmdt(str);

str = (char\*) shmat(shmid, NULL, 0);

printf("Data Read: %s\n", str);

shmdt(str); shmctl(shmid, IPC\_RMID, NULL);

return 0;

}

10. Inter-Process Communication Using Message Queue

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

struct msg { long type; char text[100]; };

int main() {

int qid = msgget(IPC\_PRIVATE, 0666|IPC\_CREAT);

struct msg m = {1, "Hello, Message Queue!"};

msgsnd(qid, &m, sizeof(m.text), 0);

printf("Message Sent: %s\n", m.text);

msgrcv(qid, &m, sizeof(m.text), 1, 0);

printf("Message Received: %s\n", m.text);

msgctl(qid, IPC\_RMID, NULL);

return 0;

}

11. Multithreading

#include <stdio.h>

#include <pthread.h>

void \*task(void \*arg) { printf("Thread %d\n", \*(int\*)arg); return NULL; }

int main() {

pthread\_t t[2]; int id[] = {1, 2};

for (int i = 0; i < 2; i++) pthread\_create(&t[i], NULL, task, &id[i]);

for (int i = 0; i < 2; i++) pthread\_join(t[i], NULL);

return 0;

}

12. Dining Philosophers Problem

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t f[5];

void \*philosopher(void \*n) {

int i = \*(int\*)n;

sem\_wait(&f[i]); sem\_wait(&f[(i+1)%5]);

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

sem\_post(&f[i]); sem\_post(&f[(i+1)%5]);

return NULL;

}

int main() {

pthread\_t t[5]; int id[] = {0, 1, 2, 3, 4};

for (int i = 0; i < 5; i++) sem\_init(&f[i], 0, 1);

for (int i = 0; i < 5; i++) pthread\_create(&t[i], NULL, philosopher, &id[i]);

for (int i = 0; i < 5; i++) pthread\_join(t[i], NULL);

return 0;

}

13. Memory Allocation Strategies

#include <stdio.h>

int main() {

int b[] = {5, 10, 20}, p[] = {10, 5, 15}, a[3] = {0};

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (!a[j] && b[j] >= p[i]) { a[j] = 1; printf("P%d -> B%d\n", i+1, j+1); break; }

return 0;

}

14. Single-Level Directory Organization

#include <stdio.h>

#include <string.h>

int main() {

char dir[5][20] = {"file1", "file2"}, f[20];

int n = 2;

printf("Enter file to search: "); scanf("%s", f);

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

if (strcmp(dir[i], f) == 0) return printf("File found\n"), 0;

printf("File not found\n");

return 0;

}

15. Two-Level Directory Organization

#include <stdio.h>

#include <string.h>

int main() {

char dir[2][5][20] = {{"file1", "file2"}, {"file3", "file4", "file5"}};

char f[20];

int d, n;

printf("Enter directory (0 or 1): "); scanf("%d", &d);

printf("Enter file to search: "); scanf("%s", f);

for (n = 0; n < 5; n++)

if (strcmp(dir[d][n], f) == 0) return printf("File found in Directory %d\n", d), 0;

printf("File not found\n");

return 0;

}

16. Random Access File Processing

#include <stdio.h>

struct emp { int id; char name[20]; };

int main() {

FILE \*f = fopen("emp.dat", "w+");

struct emp e = {1, "John"};

fseek(f, 0, SEEK\_SET); fwrite(&e, sizeof(e), 1, f);

fseek(f, 0, SEEK\_SET); fread(&e, sizeof(e), 1, f);

printf("ID: %d, Name: %s\n", e.id, e.name);

fclose(f);

return 0;

}

17. Banker’s Algorithm

#include <stdio.h>

int main() {

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

int max[3][3] = {{3, 3, 3}, {2, 2, 2}, {4, 3, 3}};

int avail[] = {3, 2, 2}, need[3][3], safe[3], count = 0, finished[3] = {0};

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

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

while (count < 3) {

int found = 0;

for (int i = 0; i < 3; i++) {

if (!finished[i]) {

int can\_allocate = 1;

for (int j = 0; j < 3; j++)

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

can\_allocate = 0;

break;

}

if (can\_allocate) {

for (int j = 0; j < 3; j++)

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

safe[count++] = i;

finished[i] = 1;

found = 1;

}

}

}

if (!found) { // If no process can be safely allocated

printf("System is in an unsafe state.\n");

return 1;

}

}

printf("Safe Sequence: ");

for (int i = 0; i < 3; i++)

printf("P%d ", safe[i]);

return 0;

}

18. Producer-Consumer Problem Using Semaphores

#include <stdio.h>

#include <semaphore.h>

#include <pthread.h>

#define MAX 5

int buffer[MAX], count = 0;

sem\_t empty, full;

void \*producer(void \*arg) {

for (int i = 0; i < 5; i++) { sem\_wait(&empty); buffer[count++] = i; sem\_post(&full); }

return NULL;

}

void \*consumer(void \*arg) {

for (int i = 0; i < 5; i++) { sem\_wait(&full); printf("Consumed: %d\n", buffer[--count]); sem\_post(&empty); }

return NULL;

}

int main() {

pthread\_t p, c;

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

pthread\_create(&p, NULL, producer, NULL); pthread\_create(&c, NULL, consumer, NULL);

pthread\_join(p, NULL); pthread\_join(c, NULL);

sem\_destroy(&empty); sem\_destroy(&full);

return 0;

}

19. Process Synchronization Using Mutex Locks

#include <stdio.h>

#include <pthread.h>

int count = 0;

pthread\_mutex\_t lock;

void \*increment(void \*arg) {

pthread\_mutex\_lock(&lock); count++; printf("Count: %d\n", count); pthread\_mutex\_unlock(&lock);

return NULL;

}

int main() {

pthread\_t t1, t2;

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

pthread\_create(&t1, NULL, increment, NULL); pthread\_create(&t2, NULL, increment, NULL);

pthread\_join(t1, NULL); pthread\_join(t2, NULL);

pthread\_mutex\_destroy(&lock);

return 0;

}

20. Reader-Writer Problem Using Semaphores

#include <stdio.h>

#include <semaphore.h>

#include <pthread.h>

int read\_count = 0;

sem\_t mutex, write\_lock;

void \*reader(void \*arg) {

sem\_wait(&mutex); read\_count++; if (read\_count == 1) sem\_wait(&write\_lock); sem\_post(&mutex);

printf("Reading\n");

sem\_wait(&mutex); read\_count--; if (read\_count == 0) sem\_post(&write\_lock); sem\_post(&mutex);

return NULL;

}

void \*writer(void \*arg) {

sem\_wait(&write\_lock); printf("Writing\n"); sem\_post(&write\_lock);

return NULL;

}

int main() {

pthread\_t r[3], w[2];

sem\_init(&mutex, 0, 1); sem\_init(&write\_lock, 0, 1);

for (int i = 0; i < 3; i++) pthread\_create(&r[i], NULL, reader, NULL);

for (int i = 0; i < 2; i++) pthread\_create(&w[i], NULL, writer, NULL);

for (int i = 0; i < 3; i++) pthread\_join(r[i], NULL);

for (int i = 0; i < 2; i++) pthread\_join(w[i], NULL);

sem\_destroy(&mutex); sem\_destroy(&write\_lock);

return 0;

}

21. Worst Fit Memory Management

#include <stdio.h>

int main() {

int b[] = {20, 10, 30}, p[] = {10, 15, 5}, n = 3, m = 3, a[3] = {0};

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

int idx = -1;

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

if (!a[j] && b[j] >= p[i] && (idx == -1 || b[j] > b[idx])) idx = j;

if (idx != -1) { a[idx] = 1; printf("P%d -> B%d\n", i + 1, idx + 1); }

else printf("P%d -> Not Allocated\n", i + 1);

}

return 0;

}

22. Best Fit Memory Management

#include <stdio.h>

int main() {

int b[] = {20, 10, 30}, p[] = {10, 15, 5}, n = 3, m = 3, a[3] = {0};

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

int idx = -1;

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

if (!a[j] && b[j] >= p[i] && (idx == -1 || b[j] < b[idx])) idx = j;

if (idx != -1) { a[idx] = 1; printf("P%d -> B%d\n", i + 1, idx + 1); }

else printf("P%d -> Not Allocated\n", i + 1);

}

return 0;

}

23. First Fit Memory Management

#include <stdio.h>

int main() {

int b[] = {20, 10, 30}, p[] = {10, 15, 5}, n = 3, m = 3, a[3] = {0};

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

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

if (!a[j] && b[j] >= p[i]) { a[j] = 1; printf("P%d -> B%d\n", i + 1, j + 1); break; }

if (!a[i]) printf("P%d -> Not Allocated\n", i + 1);

}

return 0;

}

24. UNIX System Calls for File Management

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

char buf[100];

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

write(fd, "Hello, UNIX system calls!", 25);

lseek(fd, 0, SEEK\_SET);

read(fd, buf, 25);

buf[25] = '\0';

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

close(fd);

return 0;

}

25. I/O System Calls of UNIX

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

int main() {

struct stat st;

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

struct dirent \*entry;

if (d) {

printf("Directory contents:\n");

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

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

closedir(d);

}

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

write(fd, "Hello", 5);

fcntl(fd, F\_DUPFD); // Duplicate file descriptor

fstat(fd, &st); // Get file metadata

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

close(fd);

return 0;

}

26. File Management Operations

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

char buf[100];

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

write(fd, "Hello, File Management!", 24);

lseek(fd, 0, SEEK\_SET);

read(fd, buf, 24);

buf[24] = '\0';

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

close(fd);

return 0;

}

27. Simulate ls Command

#include <stdio.h>

#include <dirent.h>

int main() {

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

struct dirent \*entry;

if (d) {

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

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

closedir(d);

}

return 0;

}

28. Simulate grep Command

#include <stdio.h>

#include <string.h>

int main() {

char line[256], \*match;

FILE \*f = fopen("file.txt", "r");

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

if ((match = strstr(line, "hello")) != NULL)

printf("%s", line);

}

fclose(f);

return 0;

}

#include <stdio.h>

#include <string.h>

int main() {

char line[256], \*match;

FILE \*f = fopen("file.txt", "r");

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

if ((match = strstr(line, "hello")) != NULL)

printf("%s", line);

}

fclose(f);

return 0;

}

29. Classical Process Synchronization Problem

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#define MAX 5

int buffer[MAX], count = 0;

sem\_t empty, full;

void \*producer(void \*arg) {

for (int i = 0; i < 5; i++) { sem\_wait(&empty); buffer[count++] = i; sem\_post(&full); }

return NULL;

}

void \*consumer(void \*arg) {

for (int i = 0; i < 5; i++) { sem\_wait(&full); printf("Consumed: %d\n", buffer[--count]); sem\_post(&empty); }

return NULL;

}

int main() {

pthread\_t p, c;

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

pthread\_create(&p, NULL, producer, NULL); pthread\_create(&c, NULL, consumer, NULL);

pthread\_join(p, NULL); pthread\_join(c, NULL);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

30. Demonstrate Thread Concepts

#include <stdio.h>

#include <pthread.h>

void \*task(void \*arg) { printf("Thread %d\n", \*(int\*)arg); return NULL; }

int main() {

pthread\_t t1, t2;

int id1 = 1, id2 = 2;

pthread\_create(&t1, NULL, task, &id1);

pthread\_create(&t2, NULL, task, &id2);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

return 0;

}

31. FIFO Paging Technique

#include <stdio.h>

int main() {

int ref[] = {1, 2, 3, 2, 1, 4, 5}, n = 7, frames[3] = {-1}, size = 3, faults = 0;

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

int hit = 0;

for (int j = 0; j < size; j++) if (frames[j] == ref[i]) hit = 1;

if (!hit) { frames[idx] = ref[i]; idx = (idx + 1) % size; faults++; }

printf("Frames: ");

for (int j = 0; j < size; j++) printf("%d ", frames[j]);

printf("\n");

}

printf("Page Faults: %d\n", faults);

return 0;

}

32. LRU Paging Technique

#include <stdio.h>

int main() {

int ref[] = {1, 2, 3, 2, 1, 4, 5}, n = 7, frames[3] = {-1}, size = 3, faults = 0, used[3] = {0};

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

int hit = 0, min = 0;

for (int j = 0; j < size; j++) if (frames[j] == ref[i]) { hit = 1; used[j] = i; break; }

if (!hit) {

for (int j = 0; j < size; j++) if (used[j] < used[min]) min = j;

frames[min] = ref[i]; used[min] = i; faults++;

}

printf("Frames: ");

for (int j = 0; j < size; j++) printf("%d ", frames[j]);

printf("\n");

}

printf("Page Faults: %d\n", faults);

return 0;

}

33. Optimal Paging Technique

#include <stdio.h>

int find\_farthest(int ref[], int n, int frames[], int size, int idx) {

int farthest = idx, victim = -1;

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

int j = idx + 1;

while (j < n && ref[j] != frames[i]) j++;

if (j > farthest) { farthest = j; victim = i; }

}

return victim == -1 ? 0 : victim;

}

int main() {

int ref[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3}, n = 12, frames[3] = {-1}, size = 3, faults = 0;

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

int hit = 0;

for (int j = 0; j < size; j++) if (frames[j] == ref[i]) hit = 1;

if (!hit) {

int victim = find\_farthest(ref, n, frames, size, i);

frames[victim] = ref[i];

faults++;

}

printf("Frames: ");

for (int j = 0; j < size; j++) printf("%d ", frames[j]);

printf("\n");

}

printf("Page Faults: %d\n", faults);

return 0;

}

34. Simulate Sequential File Allocation Strategy

#include <stdio.h>

int main() {

char records[][20] = {"Record1", "Record2", "Record3"};

for (int i = 0; i < 3; i++) {

printf("Accessing: %s\n", records[i]);

}

return 0;

}

36. Simulate Linked File Allocation Strategy

#include <stdio.h>

struct Block {

char data[20];

int next;

};

int main() {

struct Block blocks[] = {{"Block1", 1}, {"Block2", 2}, {"Block3", -1}};

int start = 0;

while (start != -1) {

printf("Accessing: %s\n", blocks[start].data);

start = blocks[start].next;

}

return 0;

}

37. FCFS Disk Scheduling

#include <stdio.h>

#include <stdlib.h>

int main() {

int req[] = {55, 58, 39, 18, 90}, n = 5, head = 50, total = 0;

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

total += abs(req[i] - head);

head = req[i];

}

printf("Total Head Movement: %d\n", total);

return 0;

}

38. SCAN Disk Scheduling

#include <stdio.h>

#include <stdlib.h>

int cmp(const void \*a, const void \*b) { return \*(int\*)a - \*(int\*)b; }

int main() {

int req[] = {55, 58, 39, 18, 90}, n = 5, head = 50, total = 0, size = 100;

qsort(req, n, sizeof(int), cmp);

printf("Head Movement: ");

for (int i = 0; i < n; i++) if (req[i] >= head) { total += abs(req[i] - head); head = req[i]; printf("%d ", req[i]); }

total += abs(size - head); head = size;

for (int i = n - 1; i >= 0; i--) if (req[i] < head) { total += abs(req[i] - head); head = req[i]; printf("%d ", req[i]); }

printf("\nTotal Head Movement: %d\n", total);

return 0;

}

39. C-SCAN Disk Scheduling

#include <stdio.h>

#include <stdlib.h>

int cmp(const void \*a, const void \*b) { return \*(int\*)a - \*(int\*)b; }

int main() {

int req[] = {55, 58, 39, 18, 90}, n = 5, head = 50, total = 0, size = 100;

qsort(req, n, sizeof(int), cmp);

printf("Head Movement: ");

for (int i = 0; i < n; i++) if (req[i] >= head) { total += abs(req[i] - head); head = req[i]; printf("%d ", req[i]); }

total += abs(size - head); head = 0;

for (int i = 0; i < n; i++) if (req[i] < head) { total += abs(req[i] - head); head = req[i]; printf("%d ", req[i]); }

printf("\nTotal Head Movement: %d\n", total);

return 0;

}

40. File Access Permissions

#include <stdio.h>

#include <sys/stat.h>

int main() {

struct stat st;

stat("file.txt", &st);

printf("File Permissions: %o\n", st.st\_mode & 0777);

return 0;

}