Complete OS Lab Programs - Simple C++ Implementations

All programs use simple, easy-to-read variable names and plain arrays. Compile on a Unix-like system for fork(), pipes, pthreads and syscalls.

1. Process Creation using fork() - Display Child PID

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

int main() {
    pid_t createdProcessID = fork();

    if (createdProcessID < 0) {
        cout << "Fork failed!" << endl;
    } else if (createdProcessID == 0) {
        cout << "Child Process ID: " << getpid() << " Parent ID: " << getppid() << endl;
    } else {
        cout << "Parent Process ID: " << getpid() << " Child ID: " << createdProcessID << endl;
    }
    return 0;
}
```

Explanation: Creates a child process with fork(). Child prints getpid() and getppid().

2. CPU Scheduling - FCFS, SJF (non-preemptive), Round Robin

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

void simulateFCFS(int processCount, int burstTime[]) {
    int waitingTime[20], turnaroundTime[20];
    waitingTime[0] = 0;
    for (int i = 1; i < processCount; i++)
        waitingTime[i] = waitingTime[i-1] + burstTime[i-1];
    int totalWT = 0, totalTAT = 0;
    for (int i = 0; i < processCount; i++) {
        turnaroundTime[i] = waitingTime[i] + burstTime[i];
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }
    cout << "FCFS Avg WT: " << (float)totalWT / processCount << " Avg TAT: " << (float)totalTAT /
processCount << endl;
}

void simulateSJF(int processCount, int burstTime[]) {
    int copyBurst[20];
    for (int i = 0; i < processCount; i++) copyBurst[i] = burstTime[i];
    for (int i = 0; i < processCount - 1; i++)
        for (int j = i + 1; j < processCount; j++)
            if (copyBurst[i] > copyBurst[j]) swap(copyBurst[i], copyBurst[j]);
    int waitingTime[20], turnaroundTime[20];
    waitingTime[0] = 0;
    for (int i = 1; i < processCount; i++) waitingTime[i] = waitingTime[i-1] + copyBurst[i-1];
```

```
    int totalWT = 0, totalTAT = 0;
    for (int i = 0; i < processCount; i++) {
        turnaroundTime[i] = waitingTime[i] + copyBurst[i];
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }
     cout << "SJF Avg WT: " << (float)totalWT / processCount << " Avg TAT: " << (float)totalTAT /
processCount << endl;
}

void simulateRoundRobin(int processCount, int burstTime[], int timeQuantum) {
    int remainingTime[20];
    for (int i = 0; i < processCount; i++) remainingTime[i] = burstTime[i];
    int waitingTime[20] = {0}, turnaroundTime[20];
    int currentTime = 0;
    bool done;
    do {
        done = true;
        for (int i = 0; i < processCount; i++) {
            if (remainingTime[i] > 0) {
                done = false;
                if (remainingTime[i] > timeQuantum) {
                    currentTime += timeQuantum;
                    remainingTime[i] -= timeQuantum;
                } else {
                    currentTime += remainingTime[i];
                    waitingTime[i] = currentTime - burstTime[i];
                    remainingTime[i] = 0;
                }
            }
        }
    } while (!done);
    int totalWT = 0, totalTAT = 0;
    for (int i = 0; i < processCount; i++) {
        turnaroundTime[i] = burstTime[i] + waitingTime[i];
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }
     cout << "RR Avg WT: " << (float)totalWT / processCount << " Avg TAT: " << (float)totalTAT /
processCount << endl;
}

int main() {
    int n;
    cout << "Enter number of processes: ";
    cin >> n;
    int burstTime[20];
    cout << "Enter burst times: ";
    for (int i = 0; i < n; i++) cin >> burstTime[i];
    simulateFCFS(n, burstTime);
    simulateSJF(n, burstTime);
    int quantum;
    cout << "Enter time quantum for RR: ";
    cin >> quantum;
    simulateRoundRobin(n, burstTime, quantum);
```

```
    return 0;
}
```
Explanation: Three scheduling simulations: FCFS, non-preemptive SJF, and Round Robin with given time quantum.

3. Inter-Process Communication using pipe()

```
#include <iostream>
#include <unistd.h>
#include <string.h>
using namespace std;

int main() {
    int pipeEnds[2];
    if (pipe(pipeEnds) == -1) { cout << "Pipe failed" << endl; return 1; }
    pid_t processID = fork();
    if (processID < 0) { cout << "Fork failed" << endl; return 1; }
    if (processID == 0) {
        close(pipeEnds[1]);
        char buffer[100];
        read(pipeEnds[0], buffer, sizeof(buffer));
        cout << "Child received: " << buffer << endl;
        close(pipeEnds[0]);
    } else {
        close(pipeEnds[0]);
        const char message[] = "Message from parent";
        write(pipeEnds[1], message, strlen(message)+1);
        close(pipeEnds[1]);
    }
    return 0;
}
```
Explanation: Parent sends a message to child through a pipe. Child reads and prints the message.

4. Process Synchronization - Producer-Consumer (using pthreads and semaphores)

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

sem_t mutexSemaphore, fullSemaphore, emptySemaphore;
int buffer[5];
int inIndex = 0, outIndex = 0;

void* producer(void*) {
    for (int i = 1; i <= 10; i++) {
        sem_wait(&emptySemaphore);
        sem_wait(&mutexSemaphore);
        buffer[inIndex] = i;
        cout << "Produced: " << i << endl;
        inIndex = (inIndex + 1) % 5;
        sem_post(&mutexSemaphore);
        sem_post(&fullSemaphore);
        sleep(1);
    }
    return NULL;
```

```
}

void* consumer(void*) {
    for (int i = 1; i <= 10; i++) {
        sem_wait(&fullSemaphore);
        sem_wait(&mutexSemaphore);
        int item = buffer[outIndex];
        cout << "Consumed: " << item << endl;
        outIndex = (outIndex + 1) % 5;
        sem_post(&mutexSemaphore);
        sem_post(&emptySemaphore);
        sleep(1);
    }
    return NULL;
}

int main() {
    pthread_t producerThread, consumerThread;
    sem_init(&mutexSemaphore, 0, 1);
    sem_init(&fullSemaphore, 0, 0);
    sem_init(&emptySemaphore, 0, 5);
    pthread_create(&producerThread, NULL, producer, NULL);
    pthread_create(&consumerThread, NULL, consumer, NULL);
    pthread_join(producerThread, NULL);
    pthread_join(consumerThread, NULL);
    return 0;
}
```

Explanation: Producer and consumer synchronize using semaphores. Buffer size is 5.

4b. Dining Philosophers (simple solution using semaphores)

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

#define PHILOSOPHER_COUNT 5
sem_t forks[PHILOSOPHER_COUNT];

void* philosopher(void* num) {
    int id = *(int*)num;
    while (true) {
        // thinking
        sleep(1);
        // pick forks (left then right)
        sem_wait(&forks[id]);
        sem_wait(&forks[(id+1)%PHILOSOPHER_COUNT]);
        cout << "Philosopher " << id << " is eating" << endl;
        sleep(1);
        sem_post(&forks[id]);
        sem_post(&forks[(id+1)%PHILOSOPHER_COUNT]);
        // done eating
        sleep(1);
        break; // single iteration for demo
```

```
    }
    return NULL;
}

int main() {
    pthread_t philosophers[PHILOSOPHER_COUNT];
    int ids[PHILOSOPHER_COUNT];
    for (int i = 0; i < PHILOSOPHER_COUNT; i++) sem_init(&forks[i], 0, 1);
     for (int i = 0; i < PHILOSOPHER_COUNT; i++) { ids[i] = i; pthread_create(&philosophers[i],
NULL, philosopher, (void*)&ids[i]); }
    for (int i = 0; i < PHILOSOPHER_COUNT; i++) pthread_join(philosophers[i], NULL);
    return 0;
}
```

Explanation: A basic dining philosophers example: each philosopher picks left then right fork; simple single-eat demo.

4c. Readers-Writers (simple solution using semaphores)

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

sem_t mutexSemaphore, writeSemaphore;
int readCount = 0;

void* reader(void*) {
    sem_wait(&mutexSemaphore);
    readCount++;
    if (readCount == 1) sem_wait(&writeSemaphore);
    sem_post(&mutexSemaphore);
    cout << "Reader is reading" << endl;
    sleep(1);
    sem_wait(&mutexSemaphore);
    readCount--;
    if (readCount == 0) sem_post(&writeSemaphore);
    sem_post(&mutexSemaphore);
    return NULL;
}

void* writer(void*) {
    sem_wait(&writeSemaphore);
    cout << "Writer is writing" << endl;
    sleep(1);
    sem_post(&writeSemaphore);
    return NULL;
}

int main() {
    pthread_t r, w;
    sem_init(&mutexSemaphore, 0, 1);
    sem_init(&writeSemaphore, 0, 1);
    pthread_create(&r, NULL, reader, NULL);
    pthread_create(&w, NULL, writer, NULL);
    pthread_join(r, NULL);
```

```
    pthread_join(w, NULL);
    return 0;
}
```

Explanation: Readers-writers solution giving preference to readers; demo with one reader and one writer.

5. Page Replacement - FIFO, LRU, Optimal

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

int simulateFIFO(int pages[], int pageCount, int frameCount) {
    int frames[10];
    for (int i = 0; i < frameCount; i++) frames[i] = -1;
    int nextReplace = 0, faults = 0;
    for (int i = 0; i < pageCount; i++) {
        bool found = false;
        for (int j = 0; j < frameCount; j++) if (frames[j] == pages[i]) found = true;
        if (!found) {
            frames[nextReplace] = pages[i];
            nextReplace = (nextReplace + 1) % frameCount;
            faults++;
        }
    }
    return faults;
}

int simulateLRU(int pages[], int pageCount, int frameCount) {
    int frames[10], timeStamp[10];
    for (int i = 0; i < frameCount; i++) { frames[i] = -1; timeStamp[i] = 0; }
    int faults = 0, currentTime = 0;
    for (int i = 0; i < pageCount; i++) {
        currentTime++;
        bool found = false;
            for (int j = 0; j < frameCount; j++) if (frames[j] == pages[i]) { found = true;
timeStamp[j] = currentTime; break; }
        if (!found) {
            int lruIndex = 0, minTime = INT_MAX;
                 for (int j = 0; j < frameCount; j++) if (timeStamp[j] < minTime) { minTime =
timeStamp[j]; lruIndex = j; }
            frames[lruIndex] = pages[i];
            timeStamp[lruIndex] = currentTime;
            faults++;
        }
    }
    return faults;
}

int simulateOptimal(int pages[], int pageCount, int frameCount) {
    int frames[10];
    for (int i = 0; i < frameCount; i++) frames[i] = -1;
    int faults = 0;
    for (int i = 0; i < pageCount; i++) {
        bool found = false;
        for (int j = 0; j < frameCount; j++) if (frames[j] == pages[i]) { found = true; break; }
```

```
        if (!found) {
            int replaceIndex = -1, farthest = i;
            for (int j = 0; j < frameCount; j++) {
                int k;
                for (k = i + 1; k < pageCount; k++) if (frames[j] == pages[k]) break;
                if (k > farthest) { farthest = k; replaceIndex = j; }
            }
            if (replaceIndex == -1) replaceIndex = 0;
            frames[replaceIndex] = pages[i];
            faults++;
        }
    }
    return faults;
}

int main() {
    int pageCount, frameCount;
    cout << "Enter number of pages: "; cin >> pageCount;
    int pages[50]; cout << "Enter reference string: "; for (int i = 0; i < pageCount; i++) cin >>
pages[i];
    cout << "Enter number of frames: "; cin >> frameCount;
    cout << "FIFO Faults: " << simulateFIFO(pages, pageCount, frameCount) << endl;
    cout << "LRU Faults: " << simulateLRU(pages, pageCount, frameCount) << endl;
    cout << "Optimal Faults: " << simulateOptimal(pages, pageCount, frameCount) << endl;
    return 0;
}
```

Explanation: Complete FIFO, LRU (timestamp), and Optimal simulations that count page faults.

6. File Allocation Methods - Contiguous, Linked, Indexed

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

void contiguousAllocation() {
    int fileCount;
    cout << "Enter number of files: "; cin >> fileCount;
    int startBlock[10], length[10];
    for (int i = 0; i < fileCount; i++) {
        cout << "Enter start block and length for file " << i+1 << ": ";
        cin >> startBlock[i] >> length[i];
    }
    cout << "Contiguous allocation mapping:\n";
    for (int i = 0; i < fileCount; i++) {
        cout << "File " << i+1 << ": ";
        for (int j = 0; j < length[i]; j++) cout << startBlock[i] + j << " ";
        cout << endl;
    }
}

void linkedAllocation() {
    int fileCount;
    cout << "Enter number of files: "; cin >> fileCount;
    int startBlock[10], length[10];
    cout << "Enter start block and length for each file:\n";
    for (int i = 0; i < fileCount; i++) cin >> startBlock[i] >> length[i];
```

```
    cout << "Linked allocation (simulated next pointers):\n";
    for (int i = 0; i < fileCount; i++) {
        int block = startBlock[i];
        cout << "File " << i+1 << ": ";
        for (int j = 0; j < length[i]; j++) { cout << block << " "; block = block + 1; }
        cout << endl;
    }
}

void indexedAllocation() {
    int fileCount;
    cout << "Enter number of files: "; cin >> fileCount;
    int indexBlock[10], length[10];
    cout << "Enter index block and length for each file:\n";
    for (int i = 0; i < fileCount; i++) cin >> indexBlock[i] >> length[i];
    cout << "Indexed allocation mapping (index block -> data blocks):\n";
    for (int i = 0; i < fileCount; i++) {
        cout << "File " << i+1 << " index " << indexBlock[i] << ": ";
        for (int j = 0; j < length[i]; j++) cout << indexBlock[i] + j + 1 << " ";
        cout << endl;
    }
}

int main() {
    cout << "Contiguous Allocation Demo\n"; contiguousAllocation();
    cout << "Linked Allocation Demo\n"; linkedAllocation();
    cout << "Indexed Allocation Demo\n"; indexedAllocation();
    return 0;
}
```

Explanation: Three simple demos showing how blocks are mapped for each allocation method.

7. Disk Scheduling - FCFS, SSTF, SCAN, CSCAN, LOOK

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

int totalHeadMovementFCFS(int requests[], int count, int head) {
    int total = 0;
    for (int i = 0; i < count; i++) { total += abs(requests[i] - head); head = requests[i]; }
    return total;
}

int totalHeadMovementSSTF(int requests[], int count, int head) {
    int done[50] = {0}, total = 0;
    for (int c = 0; c < count; c++) {
        int minDist = 1e9, index = -1;
          for (int i = 0; i < count; i++) if (!done[i] && abs(requests[i] - head) < minDist) {
minDist = abs(requests[i]-head); index = i; }
        done[index] = 1; total += minDist; head = requests[index];
    }
    return total;
}
```

```
int totalHeadMovementSCAN(int requests[], int count, int head, int diskSize) {
    int arr[60]; for (int i = 0; i < count; i++) arr[i] = requests[i];
    sort(arr, arr+count);
    int idx = 0; while (idx < count && arr[idx] < head) idx++;
    int total = 0;
    for (int i = idx; i < count; i++) { total += abs(arr[i] - head); head = arr[i]; }
    if (idx > 0) { total += abs((diskSize-1) - head); head = diskSize-1; for (int i = idx-1; i >= 0; i--) { total += abs(arr[i] - head); head = arr[i]; } }
    return total;
}

int totalHeadMovementCSCAN(int requests[], int count, int head, int diskSize) {
    int arr[60]; for (int i = 0; i < count; i++) arr[i] = requests[i];
    sort(arr, arr+count);
    int idx = 0; while (idx < count && arr[idx] < head) idx++;
    int total = 0;
    for (int i = idx; i < count; i++) { total += abs(arr[i] - head); head = arr[i]; }
    if (idx > 0) { total += abs((diskSize-1) - head); total += (diskSize-1); head = arr[0]; for (int i = 0; i < idx; i++) { total += abs(arr[i] - head); head = arr[i]; } }
    return total;
}

int totalHeadMovementLOOK(int requests[], int count, int head) {
    int arr[60]; for (int i = 0; i < count; i++) arr[i] = requests[i];
    sort(arr, arr+count);
    int idx = 0; while (idx < count && arr[idx] < head) idx++;
    int total = 0;
    for (int i = idx; i < count; i++) { total += abs(arr[i] - head); head = arr[i]; }
    for (int i = idx-1; i >= 0; i--) { total += abs(arr[i] - head); head = arr[i]; }
    return total;
}

int main() {
    int n; cout << "Enter number of requests: "; cin >> n;
    int requests[50]; for (int i = 0; i < n; i++) cin >> requests[i];
    int head; cout << "Enter initial head position: "; cin >> head;
    int diskSize = 200; // default
    cout << "FCFS Movement: " << totalHeadMovementFCFS(requests, n, head) << endl;
    cout << "SSTF Movement: " << totalHeadMovementSSTF(requests, n, head) << endl;
    cout << "SCAN Movement: " << totalHeadMovementSCAN(requests, n, head, diskSize) << endl;
    cout << "CSCAN Movement: " << totalHeadMovementCSCAN(requests, n, head, diskSize) << endl;
    cout << "LOOK Movement: " << totalHeadMovementLOOK(requests, n, head) << endl;
    return 0;
}
```

Explanation: Complete implementations for five disk scheduling algorithms; diskSize used by SCAN/CSCAN.

8. UNIX System Calls - cp (copy) and cat (display) using read/write

```
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
using namespace std;

int main() {
    int source = open("input.txt", O_RDONLY);
```

```
    if (source < 0) { cout << "Cannot open input.txt" << endl; return 1; }
    int dest = open("copy.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (dest < 0) { cout << "Cannot create copy.txt" << endl; return 1; }
    char buffer[1024];
    int bytesRead;
    while ((bytesRead = read(source, buffer, sizeof(buffer))) > 0) {
        write(dest, buffer, bytesRead);
    }
    close(source); close(dest);
    cout << "File copied to copy.txt" << endl;
    // cat: display file
    int fileToShow = open("copy.txt", O_RDONLY);
    while ((bytesRead = read(fileToShow, buffer, sizeof(buffer))) > 0) {
        write(1, buffer, bytesRead); // write to stdout
    }
    close(fileToShow);
    return 0;
}
```

Explanation: Copies input.txt to copy.txt using open/read/write, then displays contents (cat).

9. Advanced Linux Commands and Filters - Demonstration using system()

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

int main() {
    cout << "Sorting file.txt (sort)..." << endl;
    system("sort file.txt > sorted.txt");
    cout << "Searching for 'keyword' (grep)..." << endl;
    system("grep 'keyword' file.txt > found.txt");
    cout << "Counting words/lines/chars (wc)..." << endl;
    system("wc file.txt > wc_report.txt");
    cout << "Showing first 5 lines (head)..." << endl;
    system("head -n 5 file.txt > head.txt");
    cout << "Showing last 5 lines (tail)..." << endl;
    system("tail -n 5 file.txt > tail.txt");
    cout << "Commands executed; outputs are in files." << endl;
    return 0;
}
```

Explanation: Runs common shell filters via system() and stores output to files.