```
#include <stdio.h>

void fifoPageReplacement();
void lruPageReplacement();
int findLRU(int[], int);

int main() {
  char processType;
  int choice, tq = 0;

  printf("a. Non-Preemptive Scheduling\n  1. First Come First Serve\n  2. Shortest Job
First\n  3. Priority (Non-Preemptive)\n");
  printf("b. Preemptive Scheduling\n  1. Shortest Job Remaining First\n  2. Round Robin\n
3. Priority (Preemptive)\n");
  printf("c. Page Replacement\n  1. First In First Out\n  2. Least Recently Used\n");

  printf("Enter process type (a for Non-Preemptive, b for Preemptive, c for Page
Replacement): ");
  scanf(" %c", &processType);

  if (processType == 'a' || processType == 'b') {
    int n;
    printf("\nEnter number of processes: ");
    scanf("%d", &n);

    int id[n], at[n], bt[n], ct[n], tat[n], wt[n], pri[n], rbt[n];

    printf("Enter arrival times: ");
    for (int i = 0; i < n; i++) scanf("%d", &at[i]);
    printf("Enter burst times: ");
    for (int i = 0; i < n; i++) {
      scanf("%d", &bt[i]);
      rbt[i] = bt[i];
      id[i] = i + 1;
    }
    printf("Enter priorities: ");
    for (int i = 0; i < n; i++) scanf("%d", &pri[i]);

    printf("\nProcess\tAT\tBT\tPriority\n");
    for (int i = 0; i < n; i++) {
      printf("P%d\t%d\t%d\t%d\n", id[i], at[i], bt[i], pri[i]);
    }

    if (processType == 'a') {
```

```
    printf("Enter scheduling algorithm number (1-FCFS, 2-SJF, 3-Priority): ");
} else {
    printf("Enter scheduling algorithm number (1-SJRF, 2-RR, 3-Priority): ");
}
scanf("%d", &choice);

if (processType == 'b' && choice == 2) {
    printf("Enter Time Quantum: ");
    scanf("%d", &tq);
}

int time = 0, completed = 0;
int visited[n];
for (int i = 0; i < n; i++) visited[i] = 0;

if (processType == 'a') {
    switch (choice) {
        case 1:  // FCFS
            for (int i = 0; i < n; i++) {
                if (time < at[i]) time = at[i];
                ct[i] = time + bt[i];
                time = ct[i];
            }
            break;
        case 2:  // SJF
            while (completed < n) {
                int min_bt = 9999, index = -1;
                for (int i = 0; i < n; i++) {
                    if (!visited[i] && at[i] <= time && bt[i] < min_bt) {
                        min_bt = bt[i];
                        index = i;
                    }
                }
                if (index == -1) { time++; continue; }
                visited[index] = 1;
                ct[index] = time + bt[index];
                time = ct[index];
                completed++;
            }
            break;
        case 3:  // Priority (Non-Preemptive)
            while (completed < n) {
                int min_pri = 9999, index = -1;
                for (int i = 0; i < n; i++) {
```

```
                if (!visited[i] && at[i] <= time && pri[i] < min_pri) {
                    min_pri = pri[i];
                    index = i;
                }
            }
            if (index == -1) { time++; continue; }
            visited[index] = 1;
            ct[index] = time + bt[index];
            time = ct[index];
            completed++;
        }
        break;
    default:
        printf("Invalid choice!\n");
        return 0;
    }
} else if (processType == 'b') {
    switch (choice) {
    case 1:  // SJRF
        while (completed < n) {
            int shortest = -1, min_bt = 9999;
            for (int i = 0; i < n; i++) {
                if (at[i] <= time && rbt[i] > 0 && rbt[i] < min_bt) {
                    min_bt = rbt[i];
                    shortest = i;
                }
            }
            if (shortest == -1) {
                time++;
                continue;
            }
            rbt[shortest]--;
            time++;
            if (rbt[shortest] == 0) {
                ct[shortest] = time;
                completed++;
            }
        }
        break;
    case 2: { // RR
        int remaining = n;
        while (remaining > 0) {
            int executed = 0;
            for (int i = 0; i < n; i++) {
                if (rbt[i] > 0 && at[i] <= time) {
                    int execTime = (rbt[i] < tq) ? rbt[i] : tq;
                    rbt[i] -= execTime;
                    time += execTime;
                    if (rbt[i] == 0) {
                        ct[i] = time;
                        remaining--;
                    }
                    executed = 1;
                }
            }
            if (!executed) time++;
        }
        break;
    }
    case 3:  // Priority (Preemptive)
        while (completed < n) {
            int min_pri = 9999, index = -1;
            for (int i = 0; i < n; i++) {
                if (at[i] <= time && rbt[i] > 0 && pri[i] < min_pri) {
                    min_pri = pri[i];
                    index = i;
                }
            }
            if (index == -1) {
                time++;
                continue;
            }
            rbt[index]--;
            time++;
            if (rbt[index] == 0) {
                ct[index] = time;
                completed++;
            }
        }
        break;
    default:
        printf("Invalid choice!\n");
        return 0;
  }
}

float atat = 0, awt = 0;
printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
```

```c
        for (int i = 0; i < n; i++) {
            tat[i] = ct[i] - at[i];
            wt[i] = tat[i] - bt[i];
            atat += tat[i];
            awt += wt[i];
            printf("P%d\t%d\t%d\t%d\t%d\t%d\n", id[i], at[i], bt[i], ct[i], tat[i], wt[i]);
        }
        printf("\nAverage Turnaround Time = %.2f\n", atat / n);
        printf("Average Waiting Time = %.2f\n", awt / n);
    }

    else if (processType == 'c') {
        printf("Enter page replacement algorithm number (1-FIFO, 2-LRU): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                fifoPageReplacement();
                break;
            case 2:
                lruPageReplacement();
                break;
            default:
                printf("Invalid page replacement choice!\n");
        }
    } else {
        printf("Invalid process type!\n");
    }

    return 0;
}

void fifoPageReplacement() {
    int frames, pages[50], frame[10], n, i, j, k = 0, flag, fault = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    printf("Enter the 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++) frame[i] = -1;

    printf("\nPage\tFrames\n");
    for (i = 0; i < n; i++) {
        flag = 0;
        for (j = 0; j < frames; j++) {
            if (frame[j] == pages[i]) {
                flag = 1;
                break;
            }
        }
        if (flag == 0) {
            frame[k] = pages[i];
            k = (k + 1) % frames;
            fault++;
        }
        printf("%d\t", pages[i]);
        for (j = 0; j < frames; j++) {
            if (frame[j] != -1)
                printf("%d ", frame[j]);
            else
                printf("- ");
        }
        printf("\n");
    }
    printf("Total Page Faults = %d\n", fault);
}

int findLRU(int time[], int f) {
    int min = time[0], pos = 0;
    for (int i = 1; i < f; i++) {
        if (time[i] < min) {
            min = time[i];
            pos = i;
        }
    }
    return pos;
}

void lruPageReplacement() {
    int frames[10], pages[30], time[10], numPages, f, i, j, pos, faults = 0, counter = 0;
    int flag1, flag2;

    printf("Enter number of pages: ");
    scanf("%d", &numPages);
    printf("Enter the page reference string: ");
    for (i = 0; i < numPages; i++) scanf("%d", &pages[i]);
```

```
printf("Enter number of frames: ");
scanf("%d", &f);

for (i = 0; i < f; i++) {
   frames[i] = -1;
   time[i] = 0;
}

printf("\nPage\tFrames\n");
for (i = 0; i < numPages; i++) {
   flag1 = flag2 = 0;

   for (j = 0; j < f; j++) {
      if (frames[j] == pages[i]) {
         counter++;
         time[j] = counter;
         flag1 = flag2 = 1;
         break;
      }
   }

   if (flag1 == 0) {
      for (j = 0; j < f; j++) {
         if (frames[j] == -1) {
            counter++;
            faults++;
            frames[j] = pages[i];
            time[j] = counter;
            flag2 = 1;
            break;
         }
      }
   }

   if (flag2 == 0) {
      pos = findLRU(time, f);
      counter++;
      faults++;
      frames[pos] = pages[i];
      time[pos] = counter;
   }

   printf("%d\t", pages[i]);
   for (j = 0; j < f; j++) {
```

```
            if (frames[j] != -1)
                printf("%d ", frames[j]);
            else
                printf("- ");
        }
        printf("\n");
    }

    printf("Total Page Faults = %d\n", faults);
}
```