Process Scheduling…

FCFS

```c
#include<stdio.h>
typedef struct process {
    int pid;    int bt;
    int tat;    int wt;
}proc;

void proc_in(proc pr[], int n) {
    int i;
    printf("Enter Burst Time of,\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i);
        scanf("%d", &pr[i].bt);
        pr[i].pid = i;
    }
}

void display(proc pr[], int n) {
    int i;
    float awt = 0, atat = 0;
    printf("%4s%4s%4s%4s\n", "PId", "BT", "WT", "TAT");
    for (i = 0; i < n; i++) {
        printf("%4d%4d%4d%4d\n", pr[i].pid, pr[i].bt, pr[i].wt,
pr[i].tat);
        awt += pr[i].wt;
        atat += pr[i].tat;
    }
    printf("Average Waiting time: %.2f\n", awt/n);
    printf("Average Turn Around time: %.2f\n", atat/n);
}

void fcfs(proc pr[], int n) {
    int cur_t = 0, i;
    for (i = 0; i < n; i++) {
        pr[i].wt = cur_t;
        cur_t += pr[i].bt;
        pr[i].tat = cur_t;
    }
}

int main() {
    int n;
    printf("Enter No. of Processes: ");
    scanf("%d", &n);
    proc pr[n];
    proc_in(pr, n);
    fcfs(pr, n);
    display(pr, n);
    return 0;
}
```

Output

```
Enter No. of Processes: 3
Enter Burst Time of,
P0: 5
P1: 2
P2: 4
 PId  BT  WT TAT
   0   5   0   5
   1   2   5   7
   2   4   7  11
Average Waiting time: 4.00
Average Turn Around time: 7.67
```

SJF

```c
#include<stdio.h>
typedef struct process {
    int pid;    int bt;
    int tat;    int wt;
}proc;

void proc_in(proc pr[], int n) {
    int i;
    printf("Enter Burst Time of,\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i);
        scanf("%d", &pr[i].bt);
        pr[i].pid = i;
    }
}
```

```c
void display(proc pr[], int n) {
    int i;
    float awt = 0, atat = 0;
    printf("%4s%4s%4s%4s\n", "PId", "BT", "WT", "TAT");
    for (i = 0; i < n; i++) {
        printf("%4d%4d%4d%4d\n", pr[i].pid, pr[i].bt, pr[i].wt,
pr[i].tat);
        awt += pr[i].wt;
        atat += pr[i].tat;
    }
    printf("Average Waiting time: %.2f\n", awt/n);
    printf("Average Turn Around time: %.2f\n", atat/n);
}

void sjf(proc pr[], int n) {
    int cur_t = 0, i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (pr[j].bt > pr[j+1].bt) {
                proc temp = pr[j];
                pr[j] = pr[j+1];
                pr[j+1] = temp;
            }
        }
    }
    for (i = 0; i < n; i++) {
        pr[i].wt = cur_t;
        cur_t += pr[i].bt;
        pr[i].tat = cur_t;
    }
}

int main() {
    int n;
    printf("Enter No. of Processes: ");
    scanf("%d", &n);
    proc pr[n];
    proc_in(pr, n);
    sjf(pr, n);
    display(pr, n);
    return 0;
}
```

Output

```
Enter No. of Processes: 3
Enter Burst Time of,
P0: 5
P1: 2
P2: 4
 PId  BT  WT TAT
   1   2   0   2
   2   4   2   6
   0   5   6  11
Average Waiting time: 2.67
Average Turn Around time: 6.33
```

Priority

```c
#include<stdio.h>
typedef struct process {
    int pid;    int bt;     int pri;
    int tat;    int wt;
}proc;
void proc_in(proc pr[], int n) {
    int i;
    printf("Enter Burst Time and Priority of,\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i);
        scanf("%d %d", &pr[i].bt, &pr[i].pri);
        pr[i].pid = i;
    }
}

void display(proc pr[], int n) {
    int i;
    float awt = 0, atat = 0;
    printf("%4s%4s%4s%4s%4s\n", "PId", "BT", "Pri", "WT",
"TAT");
    for (i = 0; i < n; i++) {
        printf("%4d%4d%4d%4d%4d\n", pr[i].pid, pr[i].bt,
pr[i].pri, pr[i].wt, pr[i].tat);
        awt += pr[i].wt;
        atat += pr[i].tat;
    }
    printf("Average Waiting time: %.2f\n", awt/n);
    printf("Average Turn Around time: %.2f\n", atat/n);
}
```

```c
void priority(proc pr[], int n) {
    int cur_t = 0, i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (pr[j].pri > pr[j+1].pri) {
                proc temp = pr[j];
                pr[j] = pr[j+1];
                pr[j+1] = temp;
            }
        }
    }
    for (i = 0; i < n; i++) {
        pr[i].wt = cur_t;
        cur_t += pr[i].bt;
        pr[i].tat = cur_t;
    }
}

int main() {
    int n;
    printf("Enter No. of Processes: ");
    scanf("%d", &n);
    proc pr[n];
    proc_in(pr, n);
    priority(pr, n);
    display(pr, n);
    return 0;
}
```

Output

```
Enter No. of Processes: 3
Enter Burst Time and Priority of,
P0: 4 2
P1: 2 1
P2: 6 3
 PId  BT Pri  WT TAT
   1   2   1   0   2
   0   4   2   2   6
   2   6   3   6  12
Average Waiting time: 2.67
Average Turn Around time: 6.67
```

Round Robin

```c
#include<stdio.h>
typedef struct process {
    int pid;    int at;     int bt;
    int tat;    int wt;     int ct;
    int rbt;
}proc;

void proc_in(proc pr[], int n) {
    int i;
    printf("Enter Arrival Time, Burst Time of,\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i);
        scanf("%d %d",&pr[i].at, &pr[i].bt);
        pr[i].rbt = pr[i].bt;
        pr[i].pid = i;
        pr[i].ct = -1;
    }
}

void display(proc pr[], int n) {
    int i;
    float awt = 0, atat = 0;
    printf("%4s%4s%4s%4s%4s\n", "PId", "AT", "BT", "WT", "TAT");
    for (i = 0; i < n; i++) {
        printf("%4d%4d%4d%4d%4d\n", pr[i].pid, pr[i].at,
pr[i].bt, pr[i].wt, pr[i].tat);
        awt += pr[i].wt;
        atat += pr[i].tat;
    }
    printf("Average Waiting time: %.2f\n", awt/n);
    printf("Average Turn Around time: %.2f\n", atat/n);
}

void round_robin(proc pr[], int n, int tq) {
    int cur_t = 0, tm, comp = 0, wqp = 0, wqt = 0, wqn = 0,
wq[n];
    for (int i = 0; i < n; i++) {
        if (pr[i].at <= cur_t) {
            wq[wqp] = i;
            wqp = (wqp + 1)% n;
            wqn++;
        }
    }
```

```
    while (comp < n) {
        if (tq < pr[wq[wqt]].rbt)
            tm = tq;
        else
            tm = pr[wq[wqt]].rbt;

        for (int i = 0; i < n; i++) {
            if (pr[i].at <= cur_t+tm && pr[i].at > cur_t &&
pr[i].ct < 0) {
                wq[wqp] = i;
                wqp = (wqp + 1)% n;
                wqn++;
            }
        }
        if (tq < pr[wq[wqt]].rbt) {
            tm = tq;
            wq[wqp] = wqt;
            wqp = (wqp + 1)% n;
        } else {
            tm = pr[wq[wqt]].rbt;
            comp++;
            pr[wq[wqt]].ct = cur_t + tm;
            wqn--;
        }
        pr[wq[wqt]].rbt -= tm;
        wqt = (wqt + 1)% n;
        cur_t += tm;
    }
    for (int i = 0; i < n; i++) {
        pr[i].tat = pr[i].ct - pr[i].at;
        pr[i].wt = pr[i].tat - pr[i].bt;
    }
}
int main() {
    int n, tq;
    printf("Enter No. of Processes: ");
    scanf("%d", &n);
    proc pr[n];
    proc_in(pr, n);
    printf("Enter Time Quantum: ");
    scanf("%d", &tq);
    round_robin(pr, n, tq);
    display(pr, n);
    return 0;
}
```

Output

```
Enter No. of Processes: 4
Enter Arrival Time, Burst Time of,
P0: 0 6
P1: 0 4
P2: 3 5
P3: 5 2
Enter Time Quantum: 3
 PId  AT  BT  WT TAT
   0   0   6   6  12
   1   0   4  11  15
   2   3   5   9  14
   3   5   2   7   9
Average Waiting time: 8.25
Average Turn Around time: 12.50
```

Bankers Algorithm…

```
#include<stdio.h>
typedef struct process {
    int alloc[10];
    int max[10];
    int need[10];
    int exec;
} proc;

void p_in(int np, int nr, proc pr[], int avl[]) {
    for (int i = 0; i < np; i++) {
        printf("Allocation of P%d: ", i+1);
        for (int j = 0; j < nr; j++) {
            scanf("%d", &pr[i].alloc[j]);
        }
        printf("Max Need of P%d: ", i+1);
        for (int j = 0; j < nr; j++) {
            scanf("%d", &pr[i].max[j]);
            pr[i].need[j] = pr[i].max[j] - pr[i].alloc[j];
            pr[i].exec = 0;
        }
    }
    printf("Enter Resource Availability: ");
    for (int i = 0; i < nr; i++)
        scanf("%d", &avl[i]);
}
```

```c
void bankers (int np, int nr, proc pr[], int avl[]) {
    int comp = 0, dl = 0;
    int safe[np], sp = 0;
    while (comp < np && !dl) {
        dl = 1;
        for (int i = 0; i < np; i++) {
            int exe = 1;
            for (int j = 0; j < nr; j++) {
                if (pr[i].need[j] > avl[j] || pr[i].exec) {
                    exe = 0;
                    break;
                }
            }
            if (exe) {
                safe[sp++] = i+1;
                comp++;
                pr[i].exec = 1;
                dl = 0;
                for (int j = 0; j < nr; j++) {
                    avl[j] += pr[i].alloc[j];
                }
            }
        }
    }
    printf("\nNeed Matrix...\n");
    for(int i = 0; i < np; i ++) {
        printf("P%d: ", i+1);
        for(int j = 0; j < nr; j++) {
            printf("%d ", pr[i].need[j]);
        }
        printf("\n");
    }
    if (dl)
        printf("System is not in Safe State...\n");
    else {
        printf("Safe Sequence: ");
        for(int i = 0; i < np; i++) {
            printf("P%d -> ", safe[i]);
        }
        printf("Halt\n");
    }
}

int main() {
    int np, nr;
    printf("Enter No. of Processes and Resources: ");
    scanf("%d %d", &np, &nr);
    proc pr[np];
    int avl[nr];
    p_in(np, nr, pr, avl);
    bankers(np, nr, pr, avl);
    return 0;
}
```

Output

```
Enter No. of Processes and Resources: 5 3
Allocation of P1: 0
1
0
Max Need of P1: 7 5 3
Allocation of P2: 2 0 0
Max Need of P2: 3 2 2
Allocation of P3: 3 0 2
Max Need of P3: 9 0 2
Allocation of P4: 2 1 1
Max Need of P4: 2 2 2
Allocation of P5: 0 0 2
Max Need of P5: 4 3 3
Enter Resource Availability: 3 3 2

Need Matrix...
P1: 7 4 3
P2: 1 2 2
P3: 6 0 0
P4: 0 1 1
P5: 4 3 1
Safe Sequence: P2 -> P4 -> P5 -> P1 -> P3 -> Halt
```

Page Replacement Algorithms...

```c
#include<stdio.h>

int found (int f[], int w, int key) {
    for (int i = 0; i < w; i++)
        if (f[i] == key)
            return 1;
    return 0;
}
```

```c
void fifo (int rs[], int n, int w) {
    int f[w];
    for (int i = 0; i < w; i++)
        f[i] = -1;
    printf("\nImplementing FIFO...\n");
    int fp = 0, h;
    int hc = 0, fc = 0;
    printf("%10s%20s%20s\n\n", "Access", "Page in Memory",
"Fault / Hit");
    for (int i = 0; i < n; i++) {
        if (found(f, w, rs[i])) {
            hc++;
            h = 1;
        }
        else {
            f[fp] = rs[i];
            fc++;
            fp = (fp + 1)% w;
            h = 0;
        }
        char fs[2*w];
        for (int j = 0, frp = 0; j < (2*w); j+=2, frp++) {
            if (f[frp] != -1)
                fs[j] = f[frp] + '0';
            else
                fs[j] = '*';
            fs[j+1] = ' ';
        }
        if(h)
            printf("%10d%20s%20s\n", rs[i], fs, "Hit");
        else
            printf("%10d%20s%20s\n", rs[i], fs, "Falult");
    }
    printf("Hits: %d\n", hc);
    printf("Faults: %d\n", fc);
}

void lru (int rs[], int n, int w) {
    int f[w], lat[w];
    for (int i = 0; i < w; i++) {
        f[i] = -1;
        lat[i] = -1;
    }

    printf("\nImplementing LRU...\n");
    int fp = 0, h;
    int hc = 0, fc = 0;
    printf("%10s%20s%20s\n\n", "Access", "Page in Memory",
"Fault / Hit");
    for (int i = 0; i < n; i++) {
        int min = w, min_in = 0;
        if (h = found(f, w, rs[i])) {
            hc++;
            lat [h-1] = i;
        }
        else {
            for (int j = 0; j < w; j++)
                if (min > lat[j]) {
                    min = lat[j];
                    min_in = j;
                }
            f[min_in] = rs[i];
            lat[min_in] = i;
            fc++;
        }
        char fs[2*w];
        for (int j = 0, frp = 0; j < (2*w); j+=2, frp++) {
            if (f[frp] != -1)
                fs[j] = f[frp] + '0';
            else
                fs[j] = '*';
            fs[j+1] = ' ';
        }
        if(h)
            printf("%10d%20s%20s\n", rs[i], fs, "Hit");
        else
            printf("%10d%20s%20s\n", rs[i], fs, "Falult");
    }
    printf("Hits: %d\n", hc);
    printf("Faults: %d\n", fc);
}

int main() {
    int n, w;
    printf("Enter lenghth of Reference String and Frame Width:
");
    scanf("%d %d", &n, &w);
    int rs[n];
    printf("Enter reference String: ");
    for (int i = 0; i < n; i++)
```

```
        scanf("%d", &rs[i]);
    fifo(rs, n, w);
    lru(rs, n, w);
    return 0;
}
```

Output

```
Enter lenghth of Reference String and Frame Width: 10 3
Enter reference String: 1 5 3 8 5 9 8 4 2 0

Implementing FIFO...
    Access      Page in Memory         Fault / Hit

         1              1 * *              Falult
         5              1 5 *              Falult
         3              1 5 3              Falult
         8              8 5 3              Falult
         5              8 5 3                 Hit
         9              8 9 3              Falult
         8              8 9 3                 Hit
         4              8 9 4              Falult
         2              2 9 4              Falult
         0              2 0 4              Falult
Hits: 2
Faults: 8

Implementing LRU...
    Access      Page in Memory         Fault / Hit

         1              1 * *              Falult
         5              1 5 *              Falult
         3              1 5 3              Falult
         8              8 5 3              Falult
         5              8 5 3                 Hit
         9              8 9 3              Falult
         8              8 9 3                 Hit
         4              8 9 4              Falult
         2              2 9 4              Falult
         0              0 9 4              Falult
Hits: 2
Faults: 8
```


Memory Allocation…

```
#include<stdio.h>
void ffit(int proc[], int block[], int np, int nb) {
    int alloc[np];
    printf("\nImplementing First Fit...\n");
    for (int i = 0; i < np; i++) {
        alloc[i] = -1;
        for (int j = 0; j < nb; j++) {
            if (proc[i] <= block[j]) {
                block[j] -= proc[i];
                alloc[i] = j+1;
                break;
            }
        }
    }
    printf("%6s%9s%9s\n", "Pno", "PSize", "Block");
    for (int i = 0; i < np; i++) {
        if (alloc[i] < 0)
            printf("%6d%9d%9s\n", i+1, proc[i], "NA");
        else
            printf("%6d%9d%9d\n", i+1, proc[i], alloc[i]);
    }
}

void bfit(int proc[], int block[], int np, int nb) {
    int alloc[np];
    printf("\nImplementing Best Fit...\n");
    for (int i = 0; i < np; i++) {
        alloc[i] = -1;
        for (int j = 0; j < nb; j++) {
            if (alloc[i] == -1) {
                if (proc[i] <= block[j])
                    alloc[i] = j;
            }
            else
                if (proc[i] <= block[j] && block[j] <
block[alloc[i]])
                    alloc[i] = j;
        }
        if (alloc[i] != -1)
            block[alloc[i]] -= proc[i];
    }
    printf("%6s%9s%9s\n", "Pno", "PSize", "Block");
    for (int i = 0; i < np; i++) {
```

```
        if (alloc[i] < 0)
            printf("%6d%9d%9s\n", i+1, proc[i], "NA");
        else
            printf("%6d%9d%9d\n", i+1, proc[i], alloc[i]+1);
    }
}
void wfit(int proc[], int block[], int np, int nb) {
    int alloc[np];
    printf("\nImplementing Worst Fit...\n");
    for (int i = 0; i < np; i++) {
        alloc[i] = -1;
        for (int j = 0; j < nb; j++) {
            if (alloc[i] == -1) {
                if (proc[i] <= block[j])
                    alloc[i] = j;
            }
            else
                if (block[j] > block[alloc[i]])
                    alloc[i] = j;
        }
        if (alloc[i] != -1)
            block[alloc[i]] -= proc[i];
    }
    printf("%6s%9s%9s\n", "Pno", "PSize", "Block");
    for (int i = 0; i < np; i++) {
        if (alloc[i] < 0)
            printf("%6d%9d%9s\n", i+1, proc[i], "NA");
        else
            printf("%6d%9d%9d\n", i+1, proc[i], alloc[i]+1);
    }
}
int main() {
    int np, nb;
    printf("Enter No. of Processes and Memory Blocks: ");
    scanf("%d %d", &np, &nb);
    int proc[np], block[nb], probf[np], blocbf[nb], prowf[np],
blocwf[nb];
    printf("Enter Size of Process\n");
    for (int i = 0; i < np; i++) {
        printf("P%d: ", i+1);
        scanf("%d", &proc[i]);
        probf[i] = prowf[i] = proc[i];
    }
    printf("Enter Size of Block\n");
    for (int i = 0; i < nb; i++) {
        printf("B%d: ", i+1);
        scanf("%d", &block[i]);
        blocbf[i] = blocwf[i] = block[i];
    }
    ffit(proc, block, np, nb);
    bfit(probf, blocbf, np, nb);
    wfit(prowf, blocwf, np, nb);
    return 0;
}
```

Output

```
Enter No. of Processes and Memory Blocks: 5 4
Enter Size of Process
P1: 130
P2: 100
P3: 90
P4: 400
P5: 200
Enter Size of Block
B1: 200
B2: 350
B3: 140
B4: 230

Implementing First Fit...
   Pno    PSize    Block
     1      130        1
     2      100        2
     3       90        2
     4      400       NA
     5      200        4
Implementing Best Fit...
   Pno    PSize    Block
     1      130        3
     2      100        1
     3       90        1
     4      400       NA
     5      200        4
Implementing Worst Fit...
   Pno    PSize    Block
     1      130        2
     2      100        4
     3       90        2
     4      400       NA
     5      200        1
```