1) Process management using system calls: fork, exec, getpid, exit, wait, close

What it shows: parent creates child with fork(), child runs ls using execvp(), parent waits for child (wait()), demonstrates getpid() and exit(). Uses pipe() and close() briefly.

```
/* file: proc_demo.c
  Compile: gcc proc_demo.c -o proc_demo
*/
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>

int main() {
  pid_t pid;
  printf("Parent PID: %d\n", getpid());

  pid = fork();
  if (pid < 0) {
    perror("fork failed");
    return 1;
  } else if (pid == 0) {
    // child
    printf("Child PID: %d (I will execute 'ls -l')\n", getpid());
    char *args[] = {"ls", "-l", NULL};
    execvp(args[0], args);  // replaces child process image with ls
    // If exec returns, it failed:
    perror("exec failed");
    exit(1);
  } else {
```

```
    // parent
    int status;
    printf("Parent: waiting for child (pid=%d)...\n", pid);
    wait(&status);
    if (WIFEXITED(status))
      printf("Child exited with status %d\n", WEXITSTATUS(status));
    else
      printf("Child terminated abnormally\n");
  }
  return 0;
}
```

Example run / output (abridged):

```
$ gcc proc_demo.c -o proc_demo
$ ./proc_demo
Parent PID: 12345
Parent: waiting for child (pid=12346)...
Child PID: 12346 (I will execute 'ls -l')
total 8
-rw-r--r-- 1 user user  220 Nov 21 12:00 proc_demo.c
-rwxr-xr-x 1 user user  9280 Nov 21 12:02 proc_demo
Child exited with status 0
```

---

2) C programs for CPU scheduling algorithms

I'll give four short programs: **FCFS**, **SJF (non-preemptive)**, **Priority (non-preemptive)**, **Round Robin**. Each program reads simple inputs and prints waiting/turnaround times and average.

2.A FCFS (First Come First Serve)

```
/* file: fcfs.c
```

```
  Compile: gcc fcfs.c -o fcfs
*/
#include <stdio.h>
int main() {
   int n;
   printf("Enter number of processes: ");
   scanf("%d",&n);
   int bt[n], wt[n], tat[n];
   int i;
   for(i=0;i<n;i++){
      printf("Burst time of P%d: ", i+1);
      scanf("%d", &bt[i]);
   }
   wt[0]=0;
   for(i=1;i<n;i++) wt[i]=wt[i-1]+bt[i-1];
   for(i=0;i<n;i++) tat[i]=wt[i]+bt[i];
   double avwt=0, avtat=0;
   printf("\nP\tBT\tWT\tTAT\n");
   for(i=0;i<n;i++){
      printf("P%d\t%d\t%d\t%d\n", i+1, bt[i], wt[i], tat[i]);
      avwt+=wt[i]; avtat+=tat[i];
   }
   printf("Average WT=%.2f, Average TAT=%.2f\n", avwt/n, avtat/n);
   return 0;
}
```

Example:

```
Enter number of processes: 3
Burst time of P1: 5
```

Burst time of P2: 3

Burst time of P3: 1

P BT WT TAT

P1 5 0 5

P2 3 5 8

P3 1 8 9

Average WT=4.33, Average TAT=7.33

2.B SJF (Non-preemptive Shortest Job First)

```
/* file: sjf.c
  Compile: gcc sjf.c -o sjf
*/
#include <stdio.h>
#include <stdbool.h>
int main(){
   int n; printf("No. of processes: "); scanf("%d",&n);
   int bt[n], at[n], wt[n], tat[n], proc[n];
   for(int i=0;i<n;i++){ printf("Arrival time P%d: ",i+1); scanf("%d",&at[i]);
                  printf("Burst time  P%d: ",i+1); scanf("%d",&bt[i]); proc[i]=i+1;}
   int completed=0, t=0;
   for(int i=0;i<n;i++){ wt[i]=0; tat[i]=0; }
   while(completed < n){
      int idx=-1; int min_bt=1e9;
      for(int i=0;i<n;i++){
         if(at[i] <= t && bt[i]>0 && bt[i] < min_bt){
            min_bt = bt[i]; idx=i;
         }
```

```
        }
        if(idx == -1){ t++; continue; }
        // execute process idx fully
        t += bt[idx];
        tat[idx] = t - at[idx];
        wt[idx] = tat[idx] - (min_bt);
        bt[idx] = 0;
        completed++;
    }
    double awt=0, atat=0;
    printf("\nP\tAT\tTAT\tWT\n");
    for(int i=0;i<n;i++){
        printf("P%d\t%d\t%d\t%d\n", i+1, at[i], tat[i], wt[i]);
        awt += wt[i]; atat += tat[i];
    }
    printf("Avg WT=%.2f  Avg TAT=%.2f\n", awt/n, atat/n);
    return 0;
}
```

Example:

No. of processes: 3

Arrival time P1: 0

Burst time P1: 7

Arrival time P2: 2

Burst time P2: 4

Arrival time P3: 4

Burst time P3: 1

P AT TAT WT

P1 0 7 0

P2 2 5 1

P3 4 1 0

Avg WT=0.33 Avg TAT=4.33

2.C Priority Scheduling (Non-preemptive)

```
/* file: priority.c
  Compile: gcc priority.c -o priority
*/
#include <stdio.h>
int main(){
   int n; printf("Enter number of processes: "); scanf("%d",&n);
   int bt[n], priority[n], wt[n], tat[n], pid[n];
   for(int i=0;i<n;i++){
      pid[i]=i+1;
      printf("Burst time P%d: ", i+1); scanf("%d",&bt[i]);
      printf("Priority P%d (lower number = higher priority): ", i+1); scanf("%d",&priority[i]);
   }
   // sort by priority (simple selection sort)
   for(int i=0;i<n-1;i++){
      int min=i;
      for(int j=i+1;j<n;j++) if(priority[j] < priority[min]) min=j;
      // swap
      int t;
      t=priority[i]; priority[i]=priority[min]; priority[min]=t;
      t=bt[i]; bt[i]=bt[min]; bt[min]=t;
      t=pid[i]; pid[i]=pid[min]; pid[min]=t;
   }
```

```
    wt[0]=0;
    for(int i=1;i<n;i++) wt[i]=wt[i-1]+bt[i-1];
    for(int i=0;i<n;i++) tat[i]=wt[i]+bt[i];
    double awt=0, atat=0;
    printf("\nP\tPrio\tBT\tWT\tTAT\n");
    for(int i=0;i<n;i++){
        printf("P%d\t%d\t%d\t%d\t%d\n", pid[i], priority[i], bt[i], wt[i], tat[i]);
        awt+=wt[i]; atat+=tat[i];
    }
    printf("Avg WT=%.2f Avg TAT=%.2f\n", awt/n, atat/n);
    return 0;
}
```

Example:

```
Enter number of processes: 3
Burst time P1: 5
Priority P1 (lower number = higher priority): 2
Burst time P2: 3
Priority P2 (lower number = higher priority): 1
Burst time P3: 2
Priority P3 (lower number = higher priority): 3

P   Prio    BT  WT  TAT
P2  1       3   0   3
P1  2       5   3   8
P3  3       2   8   10
Avg WT=3.67 Avg TAT=7.00
```

2.D Round Robin (RR)

```
/* file: rr.c
  Compile: gcc rr.c -o rr
*/
#include <stdio.h>
int main(){
   int n, tq;
   printf("No. of processes: "); scanf("%d",&n);
   int bt[n], rem[n], wt[n], tat[n];
   for(int i=0;i<n;i++){ printf("Burst time P%d: ",i+1); scanf("%d",&bt[i]); rem[i]=bt[i]; }
   printf("Time quantum: "); scanf("%d",&tq);
   int t=0, done=0;
   for(int i=0;i<n;i++) wt[i]=0;
   while(done < n){
      int flag = 0;
      for(int i=0;i<n;i++){
         if(rem[i] > 0) {
            flag = 1;
            if(rem[i] <= tq) {
               t += rem[i];
               rem[i] = 0;
               tat[i] = t;
               wt[i] = tat[i] - bt[i];
               done++;
            } else {
               rem[i] -= tq;
               t += tq;
            }
         }
```

```
    }
    if(!flag) break;
  }
  double awt=0, atat=0;
  printf("\nP\tBT\tWT\tTAT\n");
  for(int i=0;i<n;i++){
    printf("P%d\t%d\t%d\t%d\n", i+1, bt[i], wt[i], tat[i]);
    awt += wt[i]; atat += tat[i];
  }
  printf("Avg WT=%.2f Avg TAT=%.2f\n", awt/n, atat/n);
  return 0;
}
```

Example:

No. of processes: 3

Burst time P1: 5

Burst time P2: 4

Burst time P3: 2

Time quantum: 2

P BT WT TAT

P1 5 5 10

P2 4 4 8

P3 2 2 4

Avg WT=3.67 Avg TAT=7.33

---

3) Illustrate inter-process communication (IPC) strategies

I'll give **two small examples**: a) pipe() between parent and child, b) mmap() anonymous shared memory.

3.A Pipe (parent → child)

```
/* file: ipc_pipe.c
  Compile: gcc ipc_pipe.c -o ipc_pipe
*/
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main() {
   int fd[2];
   if (pipe(fd) == -1) { perror("pipe"); return 1; }

   pid_t pid = fork();
   if (pid < 0) { perror("fork"); return 1; }
   else if (pid == 0) {
      // child: read message from parent
      close(fd[1]); // close write end
      char buf[100];
      int n = read(fd[0], buf, sizeof(buf)-1);
      if (n>0) {
         buf[n]=0;
         printf("Child received: '%s'\n", buf);
      }
      close(fd[0]);
   } else {
      // parent: write to pipe
```

```
    close(fd[0]); // close read end
    char *msg = "Hello from parent through pipe";
    write(fd[1], msg, strlen(msg));
    close(fd[1]);
    wait(NULL);
  }
  return 0;
}
```

Example output:

```
$ ./ipc_pipe
Child received: 'Hello from parent through pipe'
```

3.B Shared memory using mmap (POSIX anonymous)

```
/* file: ipc_shm.c
  Compile: gcc ipc_shm.c -o ipc_shm
*/
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>

int main() {
  // create anonymous shared memory
  char *mem = mmap(NULL, 4096, PROT_READ | PROT_WRITE,
                   MAP_SHARED | MAP_ANONYMOUS, -1, 0);
  if (mem == MAP_FAILED) { perror("mmap"); return 1; }
```

```
    pid_t pid = fork();
    if (pid == 0) {
        // child writes
        snprintf(mem, 4096, "Message from child (pid %d)", getpid());
        return 0;
    } else {
        wait(NULL);
        // parent reads the shared memory
        printf("Parent read from shared memory: %s\n", mem);
        munmap(mem, 4096);
    }
    return 0;
}
```

Example output:

Parent read from shared memory: Message from child (pid 12346)

---

4) Implement mutual exclusion by semaphores

Use POSIX unnamed semaphores (sem_init, sem_wait, sem_post) to protect critical section between two processes.

```
/* file: sem_demo.c
   Compile: gcc sem_demo.c -o sem_demo -pthread
*/
#include <stdio.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/wait.h>
```

```
int main(){
   // place semaphore in shared memory
   sem_t *sem = mmap(NULL, sizeof(sem_t), PROT_READ|PROT_WRITE,
                MAP_SHARED|MAP_ANONYMOUS, -1, 0);
   if (sem == MAP_FAILED) { perror("mmap"); return 1; }
   sem_init(sem, 1, 1); // shared between processes, initial value 1

   pid_t pid = fork();
   if (pid == 0) {
      // child
      for(int i=0;i<3;i++){
         sem_wait(sem);
         printf("Child entering critical section (%d)\n", i+1);
         sleep(1); // simulate work
         printf("Child leaving critical section (%d)\n", i+1);
         sem_post(sem);
         sleep(1);
      }
      return 0;
   } else {
      // parent
      for(int i=0;i<3;i++){
         sem_wait(sem);
         printf("Parent entering critical section (%d)\n", i+1);
         sleep(1);
         printf("Parent leaving critical section (%d)\n", i+1);
         sem_post(sem);
         sleep(1);
```

```
    }
    wait(NULL);
    sem_destroy(sem);
    munmap(sem, sizeof(sem_t));
  }
  return 0;
}
```

Example output (interleaving but no simultaneous critical sections):

```
Parent entering critical section (1)
Parent leaving critical section (1)
Child entering critical section (1)
Child leaving critical section (1)
Parent entering critical section (2)
Parent leaving critical section (2)
Child entering critical section (2)
Child leaving critical section (2)
...
```

---

5) Banker's Algorithm — avoid deadlock (safety check)

Program: Standard Banker's algorithm: reads n processes, m resources, max[][], alloc[][], calculates need and checks system safe state.

```
/* file: bankers.c
  Compile: gcc bankers.c -o bankers
*/
#include <stdio.h>
#include <stdbool.h>

int main(){
```

```
int n, m;
printf("Number of processes: "); scanf("%d",&n);
printf("Number of resource types: "); scanf("%d",&m);
int alloc[n][m], max[n][m], need[n][m];
int avail[m];
for(int i=0;i<m;i++){ printf("Available of R%d: ", i); scanf("%d",&avail[i]); }
for(int i=0;i<n;i++){
   printf("Allocation for P%d: ", i); for(int j=0;j<m;j++) scanf("%d",&alloc[i][j]);
   printf("Max for P%d: ", i); for(int j=0;j<m;j++) scanf("%d",&max[i][j]);
   for(int j=0;j<m;j++) need[i][j] = max[i][j] - alloc[i][j];
}
bool finished[n];
for(int i=0;i<n;i++) finished[i]=false;
int safeSeq[n], count=0;
while(count < n){
   bool found=false;
   for(int p=0;p<n;p++){
      if(!finished[p]){
         int j;
         for(j=0;j<m;j++)
            if(need[p][j] > avail[j]) break;
         if(j==m){
            // P can be allocated
            for(int k=0;k<m;k++) avail[k] += alloc[p][k];
            safeSeq[count++] = p;
            finished[p]=true;
            found=true;
         }
```

```
        }
      }
      if(!found) break;
    }
    if(count == n){
      printf("System is in safe state.\nSafe sequence: ");
      for(int i=0;i<n;i++) printf("P%d ", safeSeq[i]);
      printf("\n");
    } else {
      printf("System is NOT in safe state. Deadlock possible.\n");
    }
    return 0;
}
```

Example:

Number of processes: 5

Number of resource types: 3

Available of R0: 3

Available of R1: 3

Available of R2: 2

Allocation for P0: 0 1 0

Max for P0: 7 5 3

Allocation for P1: 2 0 0

Max for P1: 3 2 2

Allocation for P2: 3 0 2

Max for P2: 9 0 2

Allocation for P3: 2 1 1

Max for P3: 2 2 2

Allocation for P4: 0 0 2

Max for P4: 4 3 3

System is in safe state.

Safe sequence: P1 P3 P4 P0 P2

---

6) Deadlock Detection Algorithm (resource-allocation graph / detection)

This program implements the **deadlock detection** algorithm similar to Banker's detector: given allocation, request (need), and available, it finds processes that can finish; any remaining unfinished processes are deadlocked.

```
/* file: deadlock_detect.c
  Compile: gcc deadlock_detect.c -o deadlock_detect
*/
#include <stdio.h>
#include <stdbool.h>

int main(){
  int n, m;
  printf("Processes: "); scanf("%d",&n);
  printf("Resource types: "); scanf("%d",&m);
  int alloc[n][m], req[n][m], avail[m];
  for(int i=0;i<m;i++){ printf("Available R%d: ", i); scanf("%d",&avail[i]); }
  for(int i=0;i<n;i++){
    printf("Allocation P%d: ", i); for(int j=0;j<m;j++) scanf("%d",&alloc[i][j]);
    printf("Request P%d:    ", i); for(int j=0;j<m;j++) scanf("%d",&req[i][j]);
  }
  bool finished[n]; for(int i=0;i<n;i++) finished[i]=false;
  int work[m]; for(int j=0;j<m;j++) work[j]=avail[j];
  int count=0;
  while(count < n){
```

```
        bool progress=false;
        for(int i=0;i<n;i++){
            if(!finished[i]){
                int j;
                for(j=0;j<m;j++) if(req[i][j] > work[j]) break;
                if(j==m){
                    // can finish
                    for(int k=0;k<m;k++) work[k] += alloc[i][k];
                    finished[i]=true;
                    progress=true; count++;
                }
            }
        }
        if(!progress) break;
    }
    bool dead=false;
    for(int i=0;i<n;i++){
        if(!finished[i]){
            printf("Process P%d is DEADLOCKED\n", i);
            dead=true;
        }
    }
    if(!dead) printf("No deadlock: all processes can finish\n");
    return 0;
}
```

Example:

Processes: 3

Resource types: 2

Available R0: 0

Available R1: 1

Allocation P0: 0 1

Request P0: 0 0

Allocation P1: 2 0

Request P1: 1 0

Allocation P2: 0 1

Request P2: 1 0

Process P1 is DEADLOCKED

Process P2 is DEADLOCKED

---

7) Implement the paging technique (simple page table + FIFO page replacement)

This program simulates logical to physical address translation with fixed page size, a small page table, and demonstrates page faults handled by FIFO replacement.

```
/* file: paging_sim.c
  Compile: gcc paging_sim.c -o paging_sim
*/
#include <stdio.h>
#include <stdlib.h>

int main(){
   int logicalSize, pageSize, numPages, numFrames;
   printf("Logical address space (bytes): "); scanf("%d",&logicalSize);
   printf("Page size (bytes): "); scanf("%d",&pageSize);
   numPages = logicalSize / pageSize;
   printf("Number of logical pages = %d\n", numPages);
   printf("Number of physical frames: "); scanf("%d",&numFrames);
```

```
int pageTable[numPages];
for(int i=0;i<numPages;i++) pageTable[i] = -1; // -1 means not in memory

int frameToPage[numFrames];
for(int i=0;i<numFrames;i++) frameToPage[i] = -1;

int nextFreeFrame = 0, fifoPtr = 0, pageFaults = 0;

int nAccess;
printf("Number of logical address accesses: "); scanf("%d",&nAccess);
for(int a=0;a<nAccess;a++){
   int logicalAddr;
   printf("Access %d - logical address: ", a+1); scanf("%d",&logicalAddr);
   int page = logicalAddr / pageSize;
   int offset = logicalAddr % pageSize;
   if(page < 0 || page >= numPages){
      printf("Invalid logical address\n"); continue;
   }
   if(pageTable[page] == -1){
      // page fault
      pageFaults++;
      if(nextFreeFrame < numFrames){
         pageTable[page] = nextFreeFrame;
         frameToPage[nextFreeFrame] = page;
         nextFreeFrame++;
      } else {
         // FIFO replacement
```

```
            int victimFrame = fifoPtr;
            int victimPage = frameToPage[victimFrame];
            pageTable[victimPage] = -1;
            pageTable[page] = victimFrame;
            frameToPage[victimFrame] = page;
            fifoPtr = (fifoPtr + 1) % numFrames;
          }
          printf("Page fault occurred. Loaded page %d into frame %d\n", page,
pageTable[page]);
      } else {
          printf("Page %d already in frame %d\n", page, pageTable[page]);
      }
      int physAddr = pageTable[page]*pageSize + offset;
      printf("Translated to physical address: %d (frame %d, offset %d)\n", physAddr,
pageTable[page], offset);
    }
    printf("Total page faults = %d\n", pageFaults);
    return 0;
}
```

Example:

Logical address space (bytes): 1024

Page size (bytes): 128

Number of logical pages = 8

Number of physical frames: 3

Number of logical address accesses: 6

Access 1 - logical address: 0

Page fault occurred. Loaded page 0 into frame 0

Translated to physical address: 0 (frame 0, offset 0)

Access 2 - logical address: 300

Page fault occurred. Loaded page 2 into frame 1

Translated to physical address: 44 (frame 1, offset 44)

Access 3 - logical address: 500

Page fault occurred. Loaded page 3 into frame 2

Translated to physical address: 124 (frame 2, offset 124)

Access 4 - logical address: 0

Page 0 already in frame 0

Translated to physical address: 0 (frame 0, offset 0)

Access 5 - logical address: 700

Page fault occurred. (FIFO victim = page 0) Loaded page 5 into frame 0

Translated to physical address: 44 (frame 0, offset 44)

Access 6 - logical address: 300

Page 2 already in frame 1

Translated to physical address: 44 (frame 1, offset 44)

Total page faults = 4