```
2 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>


#define MAX 1024


void simulate_cp() {
  char src[100], dest[100];
  FILE *fp1, *fp2;
  char ch;


  printf("Enter source filename: ");
  scanf("%s", src);
  printf("Enter destination filename: ");
  scanf("%s", dest);


  fp1 = fopen(src, "r");
  if (fp1 == NULL) {
     printf("Cannot open source file.\n");
     return;
  }


  fp2 = fopen(dest, "w");
  if (fp2 == NULL) {
     printf("Cannot open/create destination file.\n");
     fclose(fp1);
     return;
```

```
  }

  while ((ch = fgetc(fp1)) != EOF) {
     fputc(ch, fp2);
  }

  printf("File copied successfully.\n");
  fclose(fp1);
  fclose(fp2);
}

void simulate_ls() {
  char dir[100];
  DIR *dp;
  struct dirent *entry;

  printf("Enter directory name: ");
  scanf("%s", dir);

  dp = opendir(dir);
  if (!dp) {
     printf("Cannot open directory.\n");
     return;
  }

  printf("Files in %s:\n", dir);
```

```
  while ((entry = readdir(dp)) != NULL)
     printf("%s\n", entry->d_name);


  closedir(dp);
}


void simulate_grep() {
  char filename[100], word[100], line[MAX];
  FILE *fp;
  int lineNum = 0, found = 0;


  printf("Enter filename: ");
  scanf("%s", filename);
  printf("Enter word to search: ");
  scanf("%s", word);


  fp = fopen(filename, "r");
  if (!fp) {
     printf("File not found.\n");
     return;
  }


while (!feof(fp)) {
  fgets(line, sizeof(line), fp);
  lineNum++;


  if (strstr(line, word)) {
```

```
      printf("%d: %s", lineNum, line);
      found = 1;
    }
  }



  if (!found)
      printf("No match found for '%s'\n", word);


  fclose(fp);
}


int main() {
  int choice;


  printf("\n UNIX COMMAND SIMULATOR \n");
  printf("1. cp (copy file)\n");
  printf("2. ls (list directory)\n");
  printf("3. grep (search word in file)\n");
  printf("Enter your choice (1/2/3): ");
  scanf("%d", &choice);


  switch (choice) {
      case 1: simulate_cp(); break;
      case 2: simulate_ls(); break;
      case 3: simulate_grep(); break;
      default: printf("Invalid choice.\n");
  }
```

```
  return 0;
}
4 #include <stdio.h>


int main() {
  int bt[20], rt[20], wt[20], tat[20], i, n, tq, time = 0, done;
  float wtavg = 0, tatavg = 0;


  // Get the number of processes
  printf("Enter the number of processes: ");
  scanf("%d", &n);


  // Get burst times for each process
  for(i = 0; i < n; i++) {
     printf("Enter Burst Time for Process %d: ", i);
     scanf("%d", &bt[i]);
     rt[i] = bt[i];  // rt = remaining time
  }


  // Get time quantum
  printf("Enter Time Quantum: ");
  scanf("%d", &tq);


  // Round Robin Scheduling
  do {
     done = 1;
     for(i = 0; i < n; i++) {
```

```
        if(rt[i] > 0) {
            done = 0;
            if(rt[i] > tq) {
                time += tq;
                rt[i] -= tq;
            } else {
                time += rt[i];
                tat[i] = time;  // Completion time = Turnaround time
                rt[i] = 0;
            }
        }
    }
} while(!done);


// Calculate waiting time and total averages
for(i = 0; i < n; i++) {
    wt[i] = tat[i] - bt[i];
    wtavg += wt[i];
    tatavg += tat[i];
}


// Output the result
printf("\n\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");
for(i = 0; i < n; i++) {
    printf("\tP%d\t\t%d\t\t%d\t\t%d\n", i, bt[i], wt[i], tat[i]);
}


// Averages
printf("\nAverage Waiting Time: %.2f", wtavg / n);
printf("\nAverage Turnaround Time: %.2f\n", tatavg / n);
```

```
  return 0;
}
```

3 Implement a simulation of the FCFS scheduling algorithm where the program should accept the number of processes, their arrival times, burst times from the user and perform the execution of these processes according to the FCFS algorithm.

```
#include <stdio.h>

int main() {
  int bt[20], wt[20], tat[20], i, n;
  float wtavg, tatavg;

  // Get the number of processes
  printf("Enter the number of processes: ");
  scanf("%d", &n);

  // Get burst times for each process
  for(i = 0; i < n; i++) {
     printf("Enter Burst Time for Process %d: ", i);
     scanf("%d", &bt[i]);
  }

  // Initialize the first process
  wt[0] = 0;  // Waiting time for the first process is always 0
  tat[0] = bt[0];  // Turnaround time for the first process is its burst time

  wtavg = 0;  // Initialize waiting time average
```

```
  tatavg = bt[0];  // Initialize turnaround time average for the first process


  // Calculate waiting times and turnaround times for the remaining processes
  for(i = 1; i < n; i++) {
     wt[i] = wt[i-1] + bt[i-1];  // Waiting time for the current process
     tat[i] = tat[i-1] + bt[i];  // Turnaround time for the current process
     wtavg += wt[i];  // Add current waiting time to the total
     tatavg += tat[i];  // Add current turnaround time to the total
  }


  // Output the results
  printf("\n\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");
  for(i = 0; i < n; i++) {
     printf("\tP%d\t\t%d\t\t%d\t\t%d\n", i, bt[i], wt[i], tat[i]);
  }


  // Calculate and display average waiting time and turnaround time
  printf("\nAverage Waiting Time: %.2f", wtavg / n);
  printf("\nAverage Turnaround Time: %.2f", tatavg / n);


  return 0;
}


I
4 #include <stdio.h>


int main() {
```

```
int bt[20], rt[20], wt[20], tat[20], i, n, tq, time = 0, done;
float wtavg = 0, tatavg = 0;



printf("Enter the number of processes: ");
scanf("%d", &n);



for(i = 0; i < n; i++) {
   printf("Enter Burst Time for Process %d: ", i);
   scanf("%d", &bt[i]);
   rt[i] = bt[i];
}



printf("Enter Time Quantum: ");
scanf("%d", &tq);



do {
   done = 1;
   for(i = 0; i < n; i++) {
      if(rt[i] > 0) {
         done = 0;
         if(rt[i] > tq) {
            time += tq;
            rt[i] -= tq;
         } else {
            time += rt[i];
```

```
                tat[i] = time;
                rt[i] = 0;
            }
        }
    }
  } while(!done);



  for(i = 0; i < n; i++) {
     wt[i] = tat[i] - bt[i];
     wtavg += wt[i];
     tatavg += tat[i];
  }



  printf("\n\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");
  for(i = 0; i < n; i++) {
     printf("\tP%d\t\t%d\t\t%d\t\t%d\n", i, bt[i], wt[i], tat[i]);
  }



  printf("\nAverage Waiting Time: %.2f", wtavg / n);
  printf("\nAverage Turnaround Time: %.2f\n", tatavg / n);


  return 0;
}
5: #include <stdio.h>
#include <stdlib.h>
```

```
int mutex = 1, full = 0, empty = 3, x = 0;


int wait(int s) { return --s; }
int signal(int s) { return ++s; }


void produce() {
  mutex = wait(mutex);
  printf("Produced %d\n", ++x);
  full = signal(full);
  empty = wait(empty);
  mutex = signal(mutex);
}


void consume() {
  mutex = wait(mutex);
  printf("Consumed %d\n", x--);
  full = wait(full);
  empty = signal(empty);
  mutex = signal(mutex);
}


int main() {
  int c;
  while (1) {
     printf("\n1.Produce  2.Consume  3.Exit\n> ");
     scanf("%d", &c);
     if (c == 1)
        (empty && mutex) ? produce() : printf("Full\n");
```

```
    else if (c == 2)
      (full && mutex) ? consume() : printf("Empty\n");
    else if (c == 3)
      break;
    else
      printf("Invalid\n");
  }
  return 0;
}
7 #include <stdio.h>


int main() {
  int nb, nf, i, j;


  printf("Enter number of blocks: ");
  scanf("%d", &nb);
  int block[nb], blockUsed[nb];  // memory blocks & usage flag


  printf("Enter sizes of %d blocks:\n", nb);
  for (i = 0; i < nb; i++) {
    printf("Block %d: ", i + 1);
    scanf("%d", &block[i]);
    blockUsed[i] = 0; // mark as unallocated
  }


  printf("Enter number of files: ");
  scanf("%d", &nf);
  int file[nf], assignedBlock[nf], fragment[nf];
```

```
  printf("Enter sizes of %d files:\n", nf);
  for (i = 0; i < nf; i++) {
     printf("File %d: ", i + 1);
     scanf("%d", &file[i]);
     assignedBlock[i] = -1; // initially unassigned


     for (j = 0; j < nb; j++) {
        if (!blockUsed[j] && block[j] >= file[i]) {
           assignedBlock[i] = j;
           fragment[i] = block[j] - file[i];
           blockUsed[j] = 1; // mark block as used
           break;
        }
     }
  }


  printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n");
  for (i = 0; i < nf; i++) {
     if (assignedBlock[i] != -1)
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, file[i], assignedBlock[i] + 1, block[assignedBlock[i]], fragment[i]);
     else
        printf("%d\t%d\t\tNot Allocated\n", i + 1, file[i]);
  }


  return 0;
}
1 #include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <sys/wait.h>
void performOperation(char op, int a, int b) {
int result;
switch (op) {
case '+': result = a + b; break;
case '-': result = a - b; break;
case '*': result = a * b; break;
case '/':
if (b == 0) {
printf("Cannot divide by 0\n");
exit(1);
}
result = a / b;
break;
default:
printf("Invalid operation\n");
exit(1);
}
printf("Result: %d\n", result);
exit(result);
}
int main() {
char op;
int a, b, status;
printf("Enter operation (+ - * /): ");
scanf(" %c", &op);
printf("Enter 2 numbers: ");
scanf("%d %d", &a, &b);
pid_t pid = fork();
if (pid == 0) {
performOperation(op, a, b);
} else {
waitpid(pid, &status, 0);
```

```
if (WIFEXITED(status)) {
printf("Child exited with result = %d\n", WEXITSTATUS(status));
}
}
return 0;
}
```