Atma Ram Sanatan Dharma College
University of Delhi

Operating Systems

Practical File for Paper Code 32341302

Submitted By
Sudipto Ghosh
College Roll No. 19/78003
BSc (Hons) Computer Science

Submitted To
Dr Parul Jain
Department of Computer Science

INDEX

14	Write a program to implement first-fit, best-fit and worst-fit allocation strategies.	
15	WAP to map logical addresses to physical addresses in a paging scheme. Define the necessary data structures required for the program. Page size and physical memory size should be taken as input from the user. Also accept process id and its size from the user and allocate memory to the process. Make an interactive program to perform the following: a. Accept a process id and page no and display frame number for a valid input b. Accept a process id to de-allocate and display the frame table.	

PRACTICAL 1

Objective

Write a program to report behaviour of Linux kernel including kernel version, CPU type and model.

Code

```
/**
 * Write a program to report behaviour of Linux kernel
 * including kernel version, CPU type and model.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  printf("Linux Kernel Version: ");
  fflush(stdout);
  system("awk 'NR == 1 {print $3;}' /proc/version");
  printf("CPU Model: ");
  fflush(stdout);
  system("awk 'NR == 5 {$1=$2=$3=\"\\b\"; print $0;}' /proc/cpuinfo");
  printf("CPU Frequency: ");
  fflush(stdout);
  system("awk 'NR == 8 {$1=$2=$3=\"\\b\"; printf $0; print \" MHz\";}' /proc/c
puinfo");
  printf("CPU Core Count: ");
  fflush(stdout);
  system("grep processor /proc/cpuinfo | wc -l");

  return 0;
}
```

Output

```
$ ./main
Linux Kernel Version: 4.19.104-microsoft-standard
CPU Model: AMD A8-7410 APU with AMD Radeon R5 Graphics
CPU Frequency: 2195.875 MHz
CPU Core Count: 4
```

PRACTICAL 2

Objective

Write a program to print file details including owner access permissions, file access time, where file name is given as argument.

Code

```
/**
 * Write a program to print file details including owner
 * access permissions, file access time,where file name
 * is given as argument.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc, char **argv)
{
  if (argc < 2)
  {
    fprintf(stderr, "Usage: ./main <file>\n");
    return -1;
  }
  struct stat buf;
  if (stat(argv[1], &buf) < 0)
  {
    fprintf(stderr, "Could not open %s\n", argv[1]);
    return 2;
  }
  printf("File Information\n------------------\n");
  printf("Name: %s\n", argv[1]);
  printf("UID: %0.4d\n", buf.st_uid);
  printf("GID: %0.4d\n", buf.st_gid);
  printf("Regular File: %s\n", S_ISREG(buf.st_mode) ? "Y" : "N");
  printf("Symbolic Link: %s\n", S_ISLNK(buf.st_mode) ? "Y" : "N");
  printf("Directory: %s\n", S_ISDIR(buf.st_mode) ? "Y" : "N");
  printf("Block Device: %s\n", S_ISBLK(buf.st_mode) ? "Y" : "N");
  printf("Character Device: %s\n", S_ISCHR(buf.st_mode) ? "Y" : "N");
  printf("File Mode Bits: %07o\n", buf.st_mode);
  printf("Last Access Time: %lld\n", buf.st_atime);
  printf("Owner Permissions:\n");
  printf("  Read: %s\n", S_IRUSR & buf.st_mode ? "Y" : "N");
  printf("  Write: %s\n", S_IWUSR & buf.st_mode ? "Y" : "N");
```

```
  printf("  Execute: %s\n", S_IXUSR & buf.st_mode ? "Y" : "N");
  printf("Group Permissions:\n");
  printf("  Read: %s\n", S_IRGRP & buf.st_mode ? "Y" : "N");
  printf("  Write: %s\n", S_IWGRP & buf.st_mode ? "Y" : "N");
  printf("  Execute: %s\n", S_IXGRP & buf.st_mode ? "Y" : "N");
  printf("Others Permissions:\n");
  printf("  Read: %s\n", S_IROTH & buf.st_mode ? "Y" : "N");
  printf("  Write: %s\n", S_IWOTH & buf.st_mode ? "Y" : "N");
  printf("  Execute: %s\n", S_IXOTH & buf.st_mode ? "Y" : "N");
  return 0;
}
```

Output

```
$ ./main output.txt
File Information
------------------
Name: output.txt
UID: 1000
GID: 1000
Regular File: Y
Symbolic Link: N
Directory: N
Block Device: N
Character Device: N
File Mode Bits: 0100777
Last Access Time: 1597132194
Owner Permissions:
  Read: Y
  Write: Y
  Execute: Y
Group Permissions:
  Read: Y
  Write: Y
  Execute: Y
Others Permissions:
  Read: Y
  Write: Y
  Execute: Y
```

PRACTICAL 3

Objective

Write a program to copy a source file into the target file and display the target file using system calls.

Code

```
/**
 * Write a program to copy a source file into the target file and
 * display the target file using system calls.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  if (argc < 3)
  {
    fprintf(stderr, "Usage: ./main <src> <dest>\n");
    return -1;
  }
  char buf;
  ssize_t bytes;
  int fdSrc, fdDest;
  mode_t wrMode = 0777;
  if ((fdSrc = open(argv[1], O_RDONLY)) < 0)
  {
    fprintf(stderr, "Could not read %s\n", argv[1]);
    return 2;
  }
  if ((fdDest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, wrMode)) < 0)
  {
    fprintf(stderr, "Could not write to %s\n", argv[2]);
    return 2;
  }
  while ((bytes = read(fdSrc, &buf, 1)) > 0)
    write(fdDest, &buf, 1);
  if (bytes < 0)
  {
    fprintf(stderr, "Could not read contents of %s\n", argv[1]);
    return 2;
  }
  if (bytes == 0)
    write(fdDest, "\n", 1);
```

```
  printf("Copied contents of %s to %s\n", argv[1], argv[2]);
  close(fdSrc);
  close(fdDest);
  if ((fdDest = open(argv[2], O_RDONLY)) < 0)
  {
    fprintf(stderr, "Could not read %s\n", argv[2]);
    return 2;
  }
  while ((bytes = read(fdSrc, &buf, 1)) > 0)
    printf("%c", buf);
  close(fdDest);
  return 0;
}
```

Output

```
$ ./main src.txt dest.txt
Copied contents of src.txt to dest.txt
This is a text file.
```

PRACTICAL 4

Objective

Write a program (using fork() and/or exec() commands) where parent and child execute: (a) same program, same code, (b) same program, different code, (c) before terminating, the parent waits for the child to finish its task.

Code

```
/**
 * Write a program (using fork() and/or exec() commands)
 * where parent and child execute:
 *  (a) same program, same code
 *  (b) same program, different code
 *  (c) before terminating, the parent waits for the child
 *      to finish its task.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

void sameProgSameCode()
{
  pid_t pidShell = getppid(), pidFork;
  printf("Shell PID: %lld\n", pidShell);
  if ((pidFork = fork()) < 0)
    fprintf(stderr, "Error in fork()");
  else
    printf("Process ID: %lld\n", getpid());
  if (pidFork == 0)
    exit(0);
  return;
}

void sameProgDiffCode()
{
  pid_t pidShell = getppid(), pidFork;
  printf("Shell PID: %lld\n", pidShell);

  if ((pidFork = fork()) < 0)
    fprintf(stderr, "Error in fork()\n");
  else if (pidFork > 0)
    printf("PARENT: Forked Child\n");
  else
```

```
  {
    printf("CHILD: Parent Process ID: %lld\n", getppid());
    printf("CHILD: Process ID: %lld\n", getpid());
    exit(0);
  }
  return;
}

void waitForChild()
{
  pid_t pidFork, pidShell = getppid();
  printf("Shell PID: %lld\n", pidShell);
  if ((pidFork = fork()) < 0)
    fprintf(stderr, "Error in fork()\n");
  else if (pidFork > 0)
  {
    wait(NULL);
    printf("PARENT: Child Exited\n");
  }
  else
  {
    printf("CHILD: Parent Process ID: %lld\n", getppid());
    printf("CHILD: Process ID: %lld\n", getpid());
    exit(0);
  }
  return;
}

int main(void)
{
  int choice;
  do
  {
    printf("=== MENU ================\n");
    printf("  (1) same program, same code\n");
    printf("  (2) same program, different code\n");
    printf("  (3) the parent waits for the child\n");
    printf("  (0) exit\n");
    printf("\nEnter Choice: ");
    scanf("%i", &choice);
    switch (choice)
    {
    case 1:
      sameProgSameCode();
      while (getchar() != '\n')
        ;
      getchar();
      system("clear");
```

```
      break;
    case 2:
      sameProgDiffCode();
      while (getchar() != '\n')
        ;
      getchar();
      system("clear");
      break;
    case 3:
      waitForChild();
      while (getchar() != '\n')
        ;
      getchar();
      system("clear");
      break;
    default:
      break;
    }
  } while (choice != 0);
  return 0;
}
```

Output

```
=== MENU ================
  (1) same program, same code
  (2) same program, different code
  (3) the parent waits for the child
  (0) exit

Enter Choice: 1
Shell PID: 11
Process ID: 55
Process ID: 56
```

```
=== MENU ================
  (1) same program, same code
  (2) same program, different code
  (3) the parent waits for the child
  (0) exit

Enter Choice: 2
Shell PID: 11
PARENT: Forked Child
CHILD: Parent Process ID: 55
CHILD: Process ID: 59
```

```
=== MENU ================
  (1) same program, same code
  (2) same program, different code
  (3) the parent waits for the child
  (0) exit

Enter Choice: 3
Shell PID: 11
PARENT: Child Exited
CHILD: Parent Process ID: 55
CHILD: Process ID: 62
```

PRACTICAL 6

Objective

Write a program to demonstrate Inter-Process Communication (IPC) between parent and child using pipe system call.

Code

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define READ_END 0
#define WRITE_END 1
#define BUFFER_SIZE 100

int main(void)
{
  int fd[2];
  pid_t pid;
  char read_msg[BUFFER_SIZE];
  char write_msg[BUFFER_SIZE] = "HELLO WORLD";
  if (pipe(fd) == -1)
  {
    fprintf(stderr, "Pipe failed");
    return 1;
  }
  if ((pid = fork()) == 0)
  {
    close(fd[WRITE_END]);
    unsigned long byteCount = read(fd[READ_END], read_msg, BUFFER_SIZE);
    printf("CHILD: %s (read %lu bytes from the pipe)\n", read_msg, byteCount);
  }
  else if (pid > 0)
  {
    close(fd[READ_END]);
    write(fd[WRITE_END], write_msg, strlen(write_msg));
    printf("PARENT: %s (wrote %lu bytes to the pipe)\n", write_msg, strlen(wri
te_msg));
  }
  else
  {
    fprintf(stderr, "error in fork()");
    return 1;
  }
  return 0;
}
```

Output

```
$ ./pipe
PARENT: HELLO WORLD (wrote 11 bytes to the pipe)
CHILD: HELLO WORLD (read 11 bytes from the pipe)
$
```

PRACTICAL 8

Objective

Write a program to implement FCFS scheduling algorithm.

Code

```
/**
 * Write a program to implement FCFS scheduling algorithm.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <stdio.h>
#include <stdlib.h>

struct process
{
  int pid;
  int burstTime;
  int arrivalTime;
  int waitingTime;
  int turnAroundTime;
};

void computeWaitingTime(struct process *processes, int processCount)
{
  processes[0].waitingTime = 0;
  for (int i = 0; i < processCount - 1; i++)
    processes[i + 1].waitingTime =
        processes[i].burstTime +
        processes[i].waitingTime;
}

void computeTurnAroundTime(struct process *processes, int processCount)
{
  for (int i = 0; i < processCount; i++)
    processes[i].turnAroundTime =
        processes[i].burstTime +
        processes[i].waitingTime -
        processes[i].arrivalTime;
}

void printAverageTimes(struct process *processes, int processCount)
{
  float totalWaitingTime = 0.0f;
  float totalTurnAroundTime = 0.0f;
  computeWaitingTime(processes, processCount);
  computeTurnAroundTime(processes, processCount);
```

```
  printf("Process ID\tBurst Time\tArrival Time\tWaiting Time\tTurn-
Around Time\n");
  printf("-------------------------------------------------------------");
  printf("-------------------------\n");
  for (int i = 0; i < processCount; i++)
  {
    totalWaitingTime += processes[i].waitingTime;
    totalTurnAroundTime += processes[i].turnAroundTime;
    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
           processes[i].pid,
           processes[i].burstTime,
           processes[i].arrivalTime,
           processes[i].waitingTime,
           processes[i].turnAroundTime);
  }
  printf("\nAverage Waiting Time = %.2f",
         totalWaitingTime / processCount);
  printf("\nAverage Turn-Around time = %.2f\n",
         totalTurnAroundTime / processCount);
}

int main(void)
{
  int processCount;
  printf("Enter Number of Processes: ");
  scanf("%i", &processCount);

  struct process processes[processCount];

  for (int i = 0; i < processCount; i++)
  {
    processes[i].pid = i + 1;
    printf("Burst Time for Process %i: ", i + 1);
    scanf("%d", &processes[i].burstTime);
    printf("Arrival Time for Process %i: ", i + 1);
    scanf("%d", &processes[i].arrivalTime);
  }

  printf("\n");

  printAverageTimes(processes, processCount);

  return 0;
}
```

Output

```
$ ./main
Enter Number of Processes: 3
Burst Time for Process 1: 10
Arrival Time for Process 1: 0
Burst Time for Process 2: 5
Arrival Time for Process 2: 0
Burst Time for Process 3: 8
Arrival Time for Process 3: 0

Process ID      Burst Time      Arrival Time    Waiting Time    Turn-Around Time
--------------------------------------------------------------------------------
1               10              0               0               10
2               5               0               10              15
3               8               0               15              23

Average Waiting Time = 8.33
Average Turn-Around time = 16.00
```

PRACTICAL 9

Objective

Write a program to implement SJF scheduling algorithm.

Code

```
/**
 * Write a program to implement SJF scheduling algorithm.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

struct process
{
  int pid;
  int burstTime;
  int arrivalTime;
  int waitingTime;
  int turnAroundTime;
};

void computeWaitingTime(struct process *processes, int processCount)
{
  int remainingTime[processCount];

  for (int i = 0; i < processCount; i++)
    remainingTime[i] = processes[i].burstTime;

  int check = 0;
  int min = INT_MAX;
  int completionTime, time = 0;
  int complete = 0, shortest = 0;

  while (complete != processCount)
  {
    for (int j = 0; j < processCount; j++)
    {
      if ((processes[j].arrivalTime <= time) &&
          (remainingTime[j] < min) && remainingTime[j] > 0)
      {
        min = remainingTime[j];
        shortest = j;
        check = 1;
      }
```

```
    }

    if (check == 0)
    {
      time++;
      continue;
    }

    remainingTime[shortest]--;

    min = remainingTime[shortest];
    if (min == 0)
      min = INT_MAX;

    if (remainingTime[shortest] == 0)
    {
      complete++;
      check = 0;
      completionTime = time + 1;

      processes[shortest].waitingTime =
          completionTime -
          processes[shortest].burstTime -
          processes[shortest].arrivalTime;

      if (processes[shortest].burstTime < 0)
        processes[shortest].burstTime = 0;
    }

    time++;
  }
}

void computeTurnAroundTime(struct process *processes, int processCount)
{
  for (int i = 0; i < processCount; i++)
    processes[i].turnAroundTime =
        processes[i].burstTime +
        processes[i].waitingTime;
}

void printAverageTimes(struct process *processes, int processCount)
{
  float totalWaitingTime = 0.0f;
  float totalTurnAroundTime = 0.0f;
  computeWaitingTime(processes, processCount);
  computeTurnAroundTime(processes, processCount);
```

```
  printf("Process ID\tBurst Time\tArrival Time\tWaiting Time\tTurn-
Around Time\n");
  printf("-------------------------------------------------------------");
  printf("-------------------------\n");
  for (int i = 0; i < processCount; i++)
  {
    totalWaitingTime += processes[i].waitingTime;
    totalTurnAroundTime += processes[i].turnAroundTime;
    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
           processes[i].pid,
           processes[i].burstTime,
           processes[i].arrivalTime,
           processes[i].waitingTime,
           processes[i].turnAroundTime);
  }
  printf("\nAverage Waiting Time = %.2f",
         totalWaitingTime / processCount);
  printf("\nAverage Turn-Around time = %.2f\n",
         totalTurnAroundTime / processCount);
}

int main(void)
{
  int processCount;

  printf("Enter Number of Processes: ");
  scanf("%i", &processCount);

  struct process processes[processCount];

  for (int i = 0; i < processCount; i++)
  {
    processes[i].pid = i + 1;
    printf("Burst Time for Process %i: ", i + 1);
    scanf("%d", &processes[i].burstTime);
    printf("Arrival Time for Process %i: ", i + 1);
    scanf("%d", &processes[i].arrivalTime);
  }

  printf("\n");

  printAverageTimes(processes, processCount);

  return 0;
}
```

Output

```
$ ./main
Enter Number of Processes: 4
Burst Time for Process 1: 6
Arrival Time for Process 1: 1
Burst Time for Process 2: 8
Arrival Time for Process 2: 1
Burst Time for Process 3: 7
Arrival Time for Process 3: 2
Burst Time for Process 4: 3
Arrival Time for Process 4: 3

Process ID      Burst Time      Arrival Time    Waiting Time    Turn-Around Time
--------------------------------------------------------------------------------
1               6               1               3               9
2               8               1               16              24
3               7               2               8               15
4               3               3               0               3

Average Waiting Time = 6.75
Average Turn-Around time = 12.75
```

PRACTICAL 11

Objective

Write a program to implement non-preemptive priority-based scheduling algorithm.

Code

```
/**
 * Write a program to implement non-preemptive priority
 * based scheduling algorithm.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <stdio.h>
#include <stdlib.h>

struct process
{
  int pid;
  int priority;
  int burstTime;
  int arrivalTime;
  int waitingTime;
  int turnAroundTime;
};

int comparisonDesc(const void *a, const void *b)
{
  return ((struct process *)a)->priority < ((struct process *)b)->priority;
}

int comparisonAsc(const void *a, const void *b)
{
  return ((struct process *)a)->pid > ((struct process *)b)->pid;
}

void computeWaitingTime(struct process *processes, int processCount)
{
  qsort(processes, processCount, sizeof(struct process), comparisonDesc);
  processes[0].waitingTime = 0;
  for (int i = 0; i < processCount - 1; i++)
    processes[i + 1].waitingTime =
        processes[i].burstTime +
        processes[i].waitingTime;
}

void computeTurnAroundTime(struct process *processes, int processCount)
{
```

```
  for (int i = 0; i < processCount; i++)
    processes[i].turnAroundTime =
        processes[i].burstTime +
        processes[i].waitingTime;
  qsort(processes, processCount, sizeof(struct process), comparisonAsc);
}

void printAverageTimes(struct process *processes, int processCount)
{
  float totalWaitingTime = 0.0f;
  float totalTurnAroundTime = 0.0f;
  computeWaitingTime(processes, processCount);
  computeTurnAroundTime(processes, processCount);
  printf("Process ID\tPriority\tBurst Time\tArrival Time\tWaiting Time\tTurn-
Around Time\n");
  printf("-----------------------------------------------------------");
  printf("----------------------------------------\n");
  for (int i = 0; i < processCount; i++)
  {
    totalWaitingTime += processes[i].waitingTime;
    totalTurnAroundTime += processes[i].turnAroundTime;
    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
           processes[i].pid,
           processes[i].priority,
           processes[i].burstTime,
           processes[i].arrivalTime,
           processes[i].waitingTime,
           processes[i].turnAroundTime);
  }
  printf("\nAverage Waiting Time = %.2f",
         totalWaitingTime / processCount);
  printf("\nAverage Turn-Around time = %.2f\n",
         totalTurnAroundTime / processCount);
}

int main(void)
{
  int processCount;

  printf("Enter Number of Processes: ");
  scanf("%i", &processCount);

  struct process processes[processCount];

  for (int i = 0; i < processCount; i++)
  {
    processes[i].pid = i + 1;
    printf("Burst Time for Process %i: ", i + 1);
```

```
    scanf("%d", &processes[i].burstTime);
    printf("Arrival Time for Process %i: ", i + 1);
    scanf("%d", &processes[i].arrivalTime);
    printf("Priority for Process %i: ", i + 1);
    scanf("%d", &processes[i].priority);
  }

  printf("\n");

  printAverageTimes(processes, processCount);

  return 0;
}
```

Output

```
$ ./main
Enter Number of Processes: 5
Burst Time for Process 1: 3
Arrival Time for Process 1: 0
Priority for Process 1: 3
Burst Time for Process 2: 5
Arrival Time for Process 2: 0
Priority for Process 2: 4
Burst Time for Process 3: 1
Arrival Time for Process 3: 0
Priority for Process 3: 1
Burst Time for Process 4: 7
Arrival Time for Process 4: 0
Priority for Process 4: 7
Burst Time for Process 5: 4
Arrival Time for Process 5: 0
Priority for Process 5: 8

Process ID      Priority        Burst Time      Arrival Time    Waiting Time    Turn-Around Time
-------------------------------------------------------------------------------------------------
1               3               3               0               16              19
2               4               5               0               11              16
3               1               1               0               19              20
4               7               7               0               4               11
5               8               4               0               0               4

Average Waiting Time = 10.00
Average Turn-Around time = 14.00
```

PRACTICAL 13

Objective

Write a program to implement Round Robin scheduling algorithm.

Code

```
/**
 * Write a program to implement Round Robin scheduling algorithm.
 *
 * Written by Sudipto Ghosh for the University of Delhi
 */

#include <stdio.h>
#include <stdlib.h>

struct process
{
  int pid;
  int burstTime;
  int arrivalTime;
  int waitingTime;
  int turnAroundTime;
};

void computeWaitingTime(struct process *processes, int processCount, int quant
um)
{
  int remainingTime[processCount];
  for (int i = 0; i < processCount; i++)
    remainingTime[i] = processes[i].burstTime;
  int time = 0;
  while (1)
  {
    int done = 1;
    for (int i = 0; i < processCount; i++)
    {
      if (remainingTime[i] > 0)
      {
        done = 0;
        if (remainingTime[i] > quantum)
        {
          time += quantum;
          remainingTime[i] -= quantum;
        }
        else
        {
          time += remainingTime[i];
          processes[i].waitingTime = time - processes[i].burstTime;
```

```
          remainingTime[i] = 0;
        }
      }
    }
    if (done == 1)
      break;
  }
}

void computeTurnAroundTime(struct process *processes, int processCount)
{
  for (int i = 0; i < processCount; i++)
    processes[i].turnAroundTime =
        processes[i].burstTime +
        processes[i].waitingTime -
        processes[i].arrivalTime;
}

void printAverageTimes(struct process *processes, int processCount, int quantu
m)
{
  float totalWaitingTime = 0.0f;
  float totalTurnAroundTime = 0.0f;
  computeWaitingTime(processes, processCount, quantum);
  computeTurnAroundTime(processes, processCount);
  printf("Process ID\tBurst Time\tArrival Time\tWaiting Time\tTurn-
Around Time\n");
  printf("-----------------------------------------------------------");
  printf("-------------------------\n");
  for (int i = 0; i < processCount; i++)
  {
    totalWaitingTime += processes[i].waitingTime;
    totalTurnAroundTime += processes[i].turnAroundTime;
    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
           processes[i].pid,
           processes[i].burstTime,
           processes[i].arrivalTime,
           processes[i].waitingTime,
           processes[i].turnAroundTime);
  }
  printf("\nAverage Waiting Time = %.2f",
         totalWaitingTime / processCount);
  printf("\nAverage Turn-Around time = %.2f\n",
         totalTurnAroundTime / processCount);
}

int main(void)
{
```

```c
  int processCount, quantum;

  printf("Enter Time Quantum: ");
  scanf("%i", &quantum);

  printf("Enter Number of Processes: ");
  scanf("%i", &processCount);

  struct process processes[processCount];

  for (int i = 0; i < processCount; i++)
  {
    processes[i].pid = i + 1;
    printf("Burst Time for Process %i: ", i + 1);
    scanf("%d", &processes[i].burstTime);
    printf("Arrival Time for Process %i: ", i + 1);
    scanf("%d", &processes[i].arrivalTime);
  }

  printf("\n");

  printAverageTimes(processes, processCount, quantum);

  return 0;
}
```

Output

```
$ ./main
Enter Time Quantum: 1
Enter Number of Processes: 3
Burst Time for Process 1: 3
Arrival Time for Process 1: 0
Burst Time for Process 2: 4
Arrival Time for Process 2: 0
3Burst Time for Process 3:
Arrival Time for Process 3: 0

Process ID      Burst Time      Arrival Time    Waiting Time    Turn-Around Time
--------------------------------------------------------------------------------
1               3               0               4               7
2               4               0               6               10
3               3               0               6               9

Average Waiting Time = 5.33
Average Turn-Around time = 8.67
```