Implementation of MOS (Multiprogramming Operating System)

First Version

Deerwalk Institute of Technology

Tribhuvan University

B. Sc. CSIT (3rd Semester)

(Concept Designed By:- Prof. Dr. Onkar Sharma, Marist College USA)

Lecturer: - Bikash Balami

Abstract

Operating system courseware can be categorized into two groups: those based on a simulator (e.g. Berkeley's Toy operating system, OSP from SUNY ect) and those that run directly on a bare machine (e.g. MINIX, XINU, Helsinki's LINUX). This project clearly falls into the first category. Bare machine OS enjoys the benefit to become intimately familiar with the low – level details of the machine architecture. Simulation based OS, on the other hand, intentionally shields the bare essentials of any particular machine architecture and allows focusing on implementing operating system concepts discussed in class or in the course text.

The purpose of this project is to reinforce the concepts of memory management, process management and I/O management that are discussed in Operating Systems. The first phase requires the design of simple machine architecture with few essential and primitive registers and control unit that fetches the user program loaded in the memory. This version, though entitled under MOS, loads one - one user program in the memory at a time for execution thus behaving as uni – programmed OS. The intension of this version is to focus on CPU hardware and its behavior (fetch – decode – execute). Though this is the preliminary stage of MOS, this version includes the user and master mode separation. This project is just an academic activity to help us know the concepts of operating systems.

The main objective of the project is to get acquainted with the fundamentals of the operating system and to some extent to have an idea of the computer architecture. The project is not to build the robust and the full fledged operating system that exists today. The project is just an academic activity to help to know the concept of the operating system.

SPECIFICAIONS OF MACHINE SEEN BY USERS

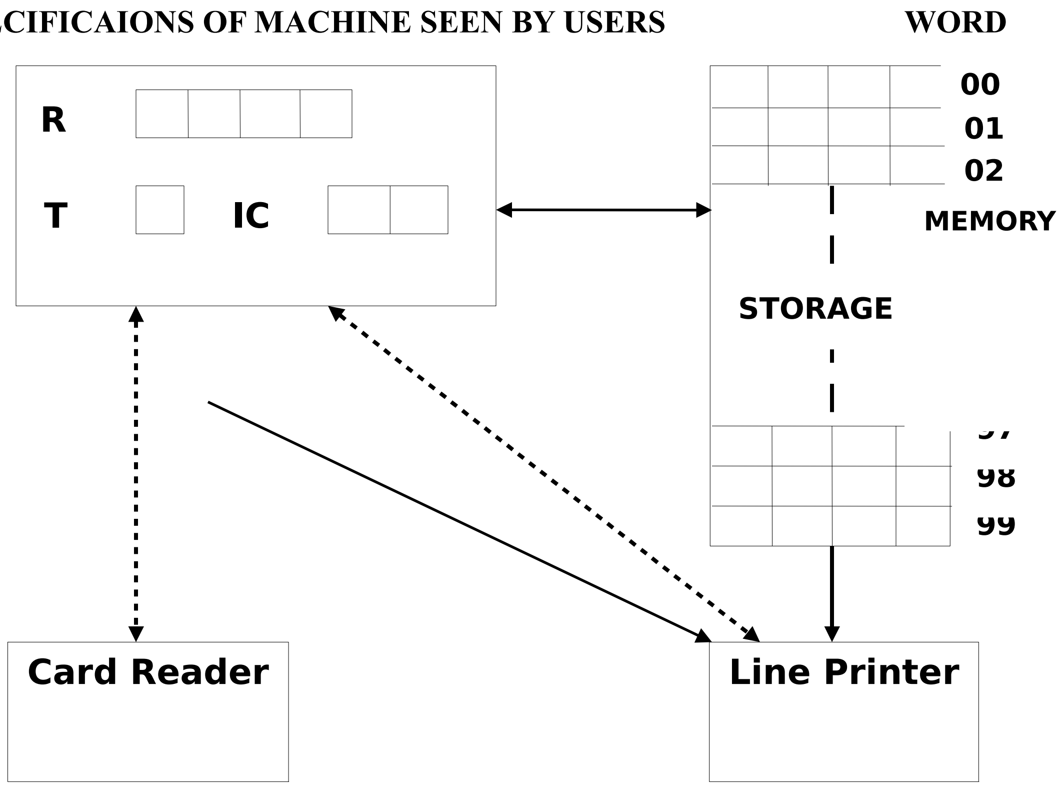

Figure 1 :- User Virtual Machine

It consists of a maximum 100 words, addressed from 00 to 99. Each word is divided into four one byte units, where a byte may contain any character acceptable by the host machine. The CPU has three registers of interest:-

a. A four byte general registers R

b. A one byte Boolean toggle C, which may contain either ‘T’ (True) or ‘F’ (False)

c. A two byte instruction counters IC

A storage word may be interpreted as an instruction or data word. The operation code of an instruction occupies the two high order bytes of the word, and the operand address appears in the two lower order bytes which are shown below.

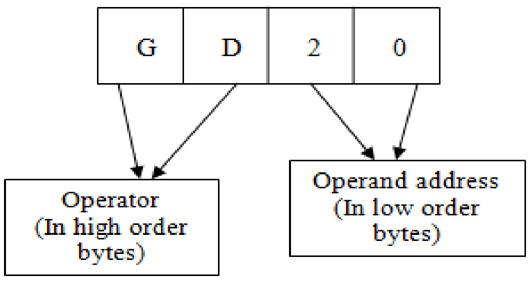
G
D
2
0
Operator (In high order bytes)
Operand address (In low order bytes)

Fig 2: Instruction format

The input instruction (GD) reads only the first 40 columns of a card and the output instruction (PD) prints the new line of 40 characters. The first instruction of a program must always appear in location 00.

<u>Job, Program and Data card formats</u>

A user job is submitted as deck of control, program and data cards in the following order:

⊼ JOB card⊣

⊼ Program cards⊣

⊼ DATA card⊣

⊼ Data⊣

⊼ ENDJOB card⊣

Where

1. The ⊼ JOB card⊣ contains four entries
   a. $AMJ, a multiprogramming job
   b. ⊼ job id⊣, a unique 4 character job identifier
   c. ⊼ time estimate⊣, 4 digit maximum time estimate to complete the job
   d. ⊼ line estimate⊣, 4 digit maximum output estimate
2. ⊼ Program card⊣ contains the information in card columns 1 – 40.
3. ⊼ DATA card⊣ has the format $DTA
4. ⊼ Data⊣ deck contains the user data retrieved by the virtual machine GD instruction.
5. ⊼ ENDJOB card⊣ has the format $END ⊼ job id⊣

<u>Sample Input Programs</u>

`$AMJ050100300008` $\longrightarrow$ *Start of the Program*

```
GD30PD30GD40LR42SR50LR41SR51PD50LR43SR50
PD50LR44SR50PD50LR40SR51PD50LR43SR50PD50
LR42SR50PD50GD60PD60H
```

$\left.\right\}$ *Program Code*

```
$DTA  ──────►  Data card started
Simulating Stack   ┐
POP PUSH1   2   3  ├ Data Lines
CALL ME LIFOFILO   ┘
$END0501  ──────►  End of the Program
```

<u>MOS CPU Instruction Set</u>

LR	≡	Load a virtual memory location's contents into R
SR	≡	Store contents of R into virtual memory location
CR	≡	Compare R to contents of virtual memory location
BT	≡	Branch on TRUE
GD	≡	Get data
PD	≡	Put data
H	≡	Halt user program

Note: All MOS instructions occupy one word of storage. The operand for the instruction, if any, occupies the last two bytes of the word.

Assumptions Made

1. Jobs entered without errors in input file in correct format, i.e. jobs error is not reported to the programmer.
2. No physical separation between jobs. Jobs are entered in the input file ("input.txt") in the batch job fashion. Here batch job means that the input jobs are in contiguous position, i.e. one after another.
3. Jobs outputs are separated in output file by 2 blank lines.
4. Program loaded in memory starting at location 00. It is assumed that only one program resides in the main memory. So the memory mapping does not include the relocation

Notations

M	:	Memory
IR	:	Instruction Register (4 bytes)
IR [1, 2]	:	Bytes 1, 2 of IR is operation code
IR [3, 4]	:	Bytes 3, 4 of IR is operand address
M [&]	:	Content of memory location &
IC	:	Instruction Counter Register (2 bytes)
R	:	General Purpose Register (4 bytes)

C	:	Toggle (1 Byte)
←	:	Loaded / Stored / Placed into

Algorithm

```
BEGIN
    INITIALIZATION
    SI = 3
    MOS (Master Mode)
        Case SI of
            1. READ
            2. WRITE
            3. TERMINATE
        End Case
    READ
        IR [4] ← 0
        Read next (data) card from input file in memory locations IR [3, 4]
        through IR [3, 4] + 9
        EXECUTEUSERPROGRAM
    WRITE
        IR [4] ← 0
        Write one block (10 words of memory) of data from memory locations
        IR[3, 4] through IR [3, 4] + 9 to the output file
        EXECUTEUSERPROGRAM
    TERMINATE
        Write 2 blank lines in output file
        LOAD
    LOAD
        M ← 0
        While not eof
            Read next (Program, Control) card from the input file in a buffer
            Control card : $AMJ, End While
                           $DTA STARTEXECUTION
                           $END, End While
```

```
                Program card : if M = 100, abort (memory exceeded)
                               Store buffer in memory location M through M + 9
                               M ← M + 9
        End While
        STOP

    STARTEXECUTION
        IC ← 00
        EXECUTEUSERPROGRAM
    END (MOS)

    EXECUTEUSERPROGRAM (SLAVE MODE)
        Loop
            IR ← M [IC]
            IC ← IC + 1
            Examine IR [1, 2]
                LR :    R ← M [IR [3, 4]]
                SR :    R → M [IR [3, 4]]
                CR :    Compare R and M [IR [3, 4]]
                            If equal C ← T else C ← F
                BT :    If C  = T then IC ← IR [3, 4]
                GD :    SI = 1 (Input Request)
                PD :    SI = 2 (Output Request)
                H :     SI = 3 (Halt)
            End Examine
        End Loop
```

Note:

EXECUTEUSER program is a slave mode function. The instructions are fetched from memory to register one by one then are compared. The first two characters of the register is checked and the following will be the cases.

Case of LR

Loads The Register R with the content of memory location specified IR [3, 4] to register R.

Case of SR

Stores The Content Of The Register R to the memory location specified by IR[3,4]

Case of CR

Compares The Contents Of Register R and the given memory location IR [3, 4]. If they are equal then the values of C is assigned as TRUE (T) or else assigned to FALSE (F).

Case of BT

Branches On TRUE, branches to memory location IR [3, 4] if C == 'T' else continues with next contiguous location IR [3, 4] + 1

Case of GD

It is **input request** so it calls the master function (supervisor mode i.e. system call) by passing the value of SI = 1. Afterwards it calls the READ function.

Case of PD

It is **output request** so it calls the master function (supervisor mode i.e. system call) by passing the value of SI = 2. Afterwards it calls the WRITE function.

Case of H

It is **terminate request** so it calls the master function (supervisor mode i.e. system call) by passing the value of SI = 3.

Sample Example

```
$AMJ050100300008
GD30PD30GD40LR42SR50LR41SR51PD50LR43SR50
PD50LR44SR50PD50LR40SR51PD50LR43SR50PD50
LR42SR50PD50GD60PD60H
$DTA
Simulating Stack
POP PUSH1   2   3
CALL ME LIFOFILO
$END0501
```

Output

```
Simulating Stack
1    PUSH
2    PUSH
3    PUSH
3    POP
2    POP
1    POP
CALL ME LIFOFILO
```

This illustration will be explained by instructor in lecture hour.

Note to the students:

The due date for submission of this project is 2013 March 8. The reports with late submission are not accepted. Students have to make their own group with upper bound of four students. If they don't arrange themselves, the instructor will take this responsibility. After finishing the simulating the first version, the individual group have to submit the documentation and have to give the presentation. Each group will get no more than 15 minutes for their presentation so arrange your time. The practical marks weight on presentation, documentation and your program. **DO NOT COPY THE CODE**. It is strictly prohibited. Wish you a best of luck.

Have Fun While Doing This Project!!!