COMP 4735: Operating Systems Concepts

Lecture 2: What is an Operating System?

Rob Neilson

rneilson@bcit.ca

Reading

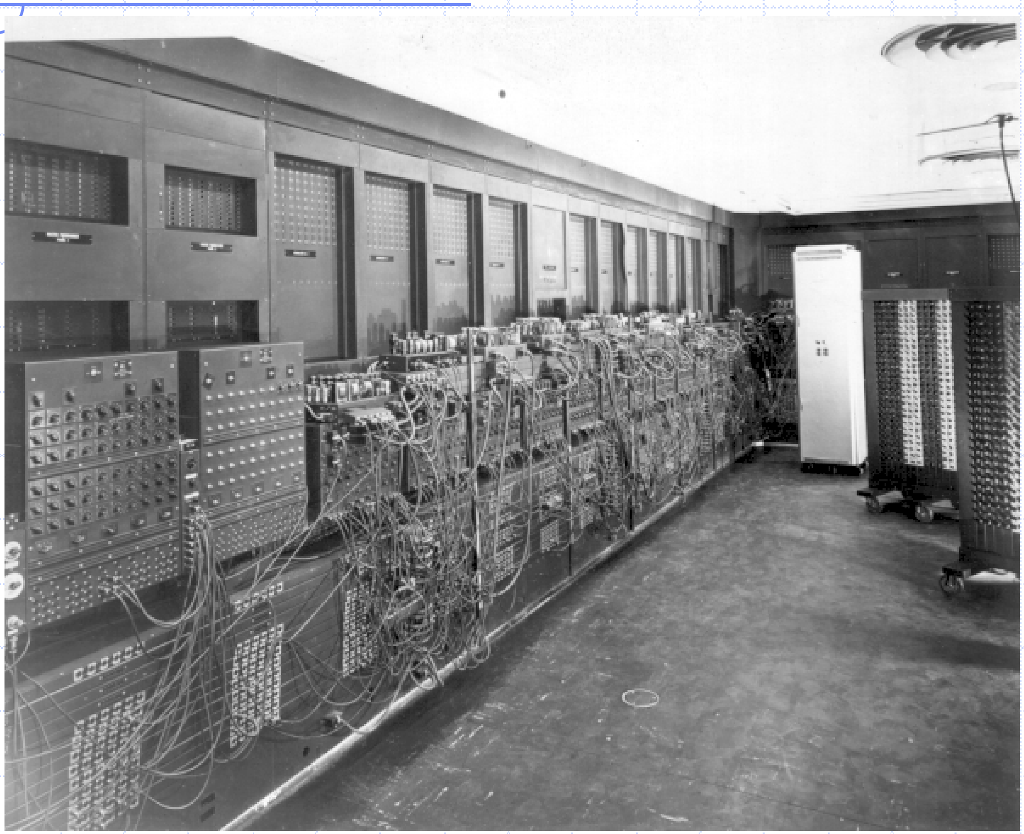
- The following sections should be read before next Monday
  - Textbook Sections: 1.1, 1.2, 1.3 (Theory Part)
    10.1, 11.1 (Case Study Part)
- Yes, there will be a quiz next Monday in lecture
  - A sample quiz will be posted on webct on Friday
  - The sample quiz will be reviewed next Monday in lecture, before the real quiz
  - This quiz (Quiz 1) covers material from all 5 of the sections listed above
    - You might do well to focus on the ‘key concepts’ presented in lecture

Typical Computer

- A typical modern computer consists of (minimally):
  - One or more processors
  - Main memory (volatile storage)
  - Disks (permanent storage)
  - Various input/output devices
- Managing all these components requires a layer of software …

  … the **operating system** …

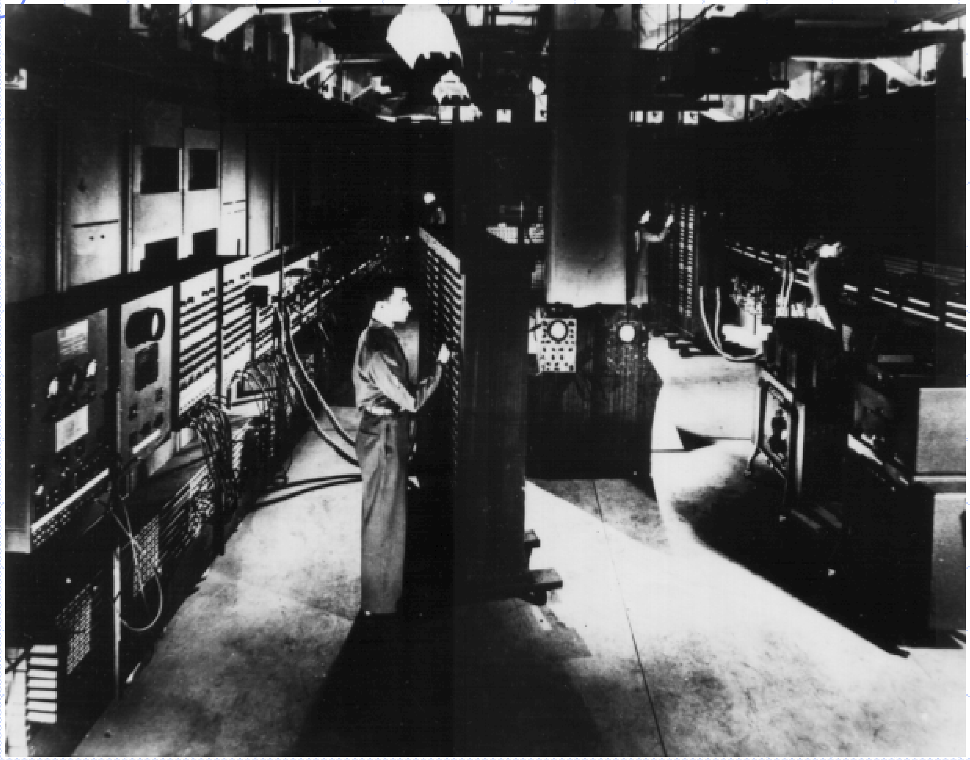
ENIAC (1946): A Typical Computer?



- thirty separate units
- 19,000 vacuum tubes
- 1,500 relays
- hundreds of thousands of resistors, capacitors, and inductors
- required 200 kilowatts of electrical power
- needed forced-air cooling
- weight: 30 tons

- ENIAC could:
  - discriminate the sign of a number
  - add, subtract, multiply, divide
  - compare quantities for equality
  - extract square roots
  - ENIAC stored a maximum of twenty 10-digit decimal numbers.

ENIAC Operating System



Which OS?
... here he is!

- ENIAC did not have an Operating System.
- Programs were ‘loaded’ by plugging tubes and relays together using cables

Life kinda sucked for ENIAC programmers ...

- ENIAC could only run one program at a time
- it had no Operating System per se
- the programmers had to move every byte or bit manually in and out of registers, accumulators etc as needed
- they plugged cables and flipped switches to set an initial state, and then turned it on
- there was no memory

"The ENIAC performed arithmetic and transfer operations simultaneously, but ***concurrent operation caused programming difficulties***."

- Martin H. Weik, 1961 Ordnance Ballistic Research Laboratories

What was the problem ...

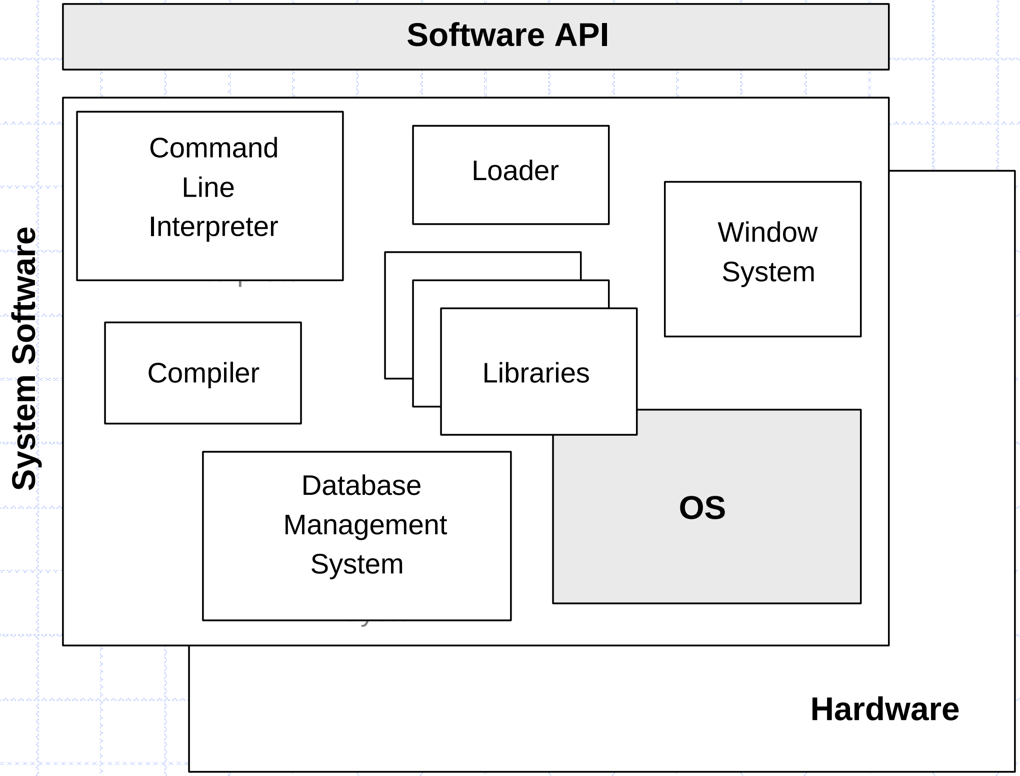
- The thing needed a better way to control it

  (actually, it needed a lot of things, but from a programmers perspective an OS would have helped)

- So what would an OS have done for them?

1. provide an abstract interface to all that hardware (***resource abstraction***)
2. allow resources to be shared (***resource sharing, or multiplexing***)
   - they got into trouble when I/O and processing (CPU) had to be done concurrently
   - what they needed was a ***super-smart dude*** who knew which wires could be plugged into which hole such that concurrent operation worked
     - this dude would be some kinda “supervisor”, or, in a modern computer, the dude is the OS

The Operating System and System Software



- In our study of operating systems, we care about the two grey boxes …
  - the Software API
    - provides an programming interface to the OS
  - the OS itself
    - abstracts and controls the hardware and resources

- The main idea is that all this stuff is the plumbing that you need to write programs – applications – to make the computer do something.

Key Concept: Resource Abstraction

- One fundamental thing any OS must do is to provide resource abstraction
- What exactly do we mean when we say “resource abstraction”?
- Well, the resources are things like disk space, cpu time, registers, etc – but we want to perform higher level ops like “write Hello World to the console”

```
load(…);
seek(…);
out(…);
```

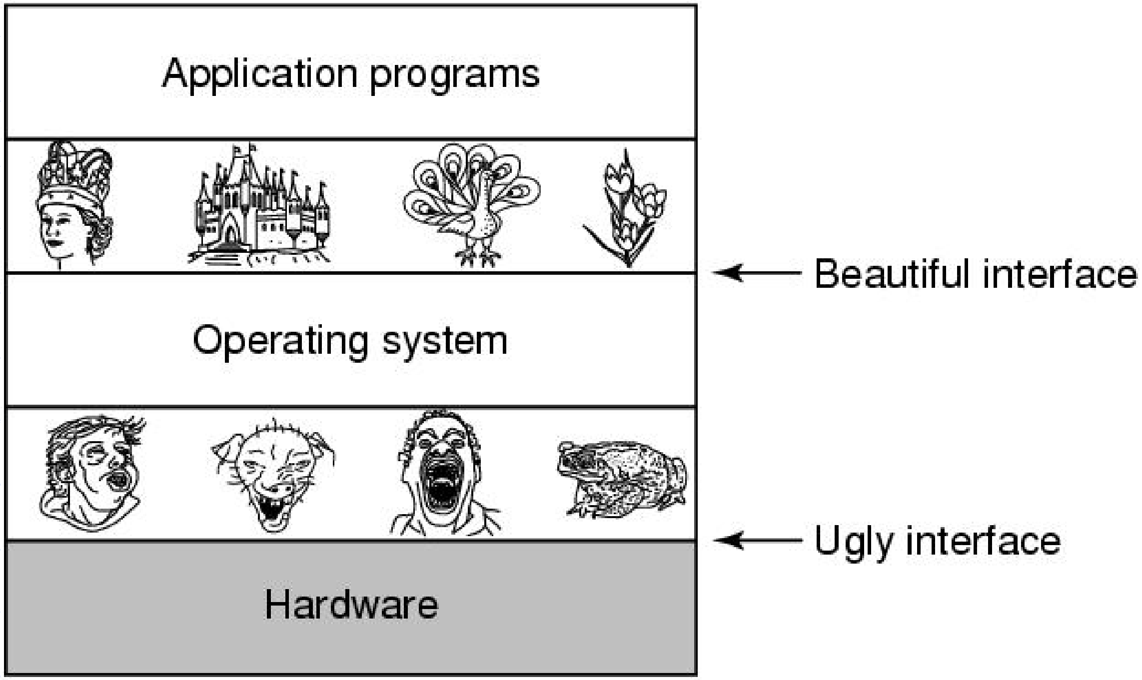
```
void write() {
   load(…);
   seek(…)
   out(…)
}
```

```
int fprintf(…) {
   ...
   write(…)
   …
}
```

- The diagrams above show progressive levels of abstraction …
  – on the left we have basic disk drive control cmds
  – on the right we have higher-level programming API calls (stdio.h)
- Each level hides details from lower levels, but …

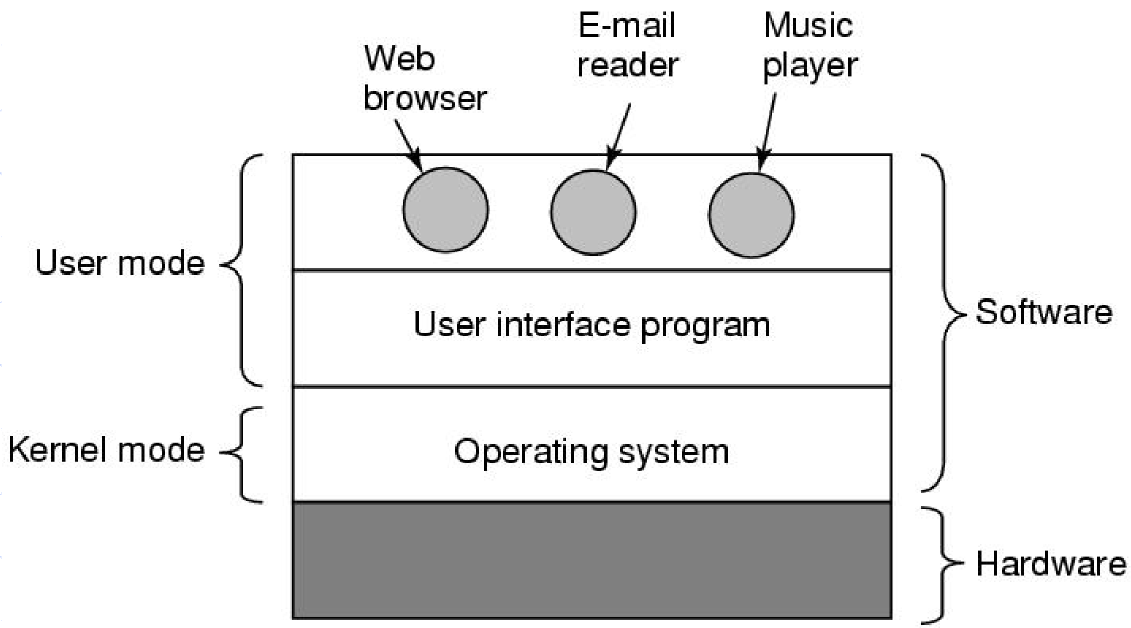
  … the trade-off is a loss in flexibility
- *Can you guess what flexibility might have been lost in the example above?*

OS vs. Hardware Interfaces



- Hardware interfaces are complex and ugly
- The OS needs to hide the device specific details from the programmer.

Where the OS fits in ...



- A modern OS uses the concepts of ***user mode*** and ***kernel mode***
- User mode software is stuff that the user or programmer might want to control
- Kernel mode stuff is critical for the operation of the computer .... we don't let the users play with it because they may cause the computer to crash

Key Concept: Resource Sharing

- The OS is basically a “resource manager”. This is its main purpose.
  - resources are things like memory, disk space, CPU

- The OS is required to:
  - Allow multiple programs to run at the same time
  - Manage and protect memory, I/O devices, and other resources

  - Includes multiplexing (sharing) resources in two different ways:
    - In time
    - In space

The OS as a Resource Manager

- Let's consider a real world example...
  - The following pages are from a book titled "DMS SuperNode System Description"
    - This is the OS used in older Nortel digital phone switches
  - These pages are the contents of one chapter that describes the OS nucleus – or kernel.
  - Notice that the OS kernel is basically one big resource manager
  - Look at the types of things this OS manages ...
    - Memory, processes, timers, queues, messages, etc

<Insert DNS PDF files here>

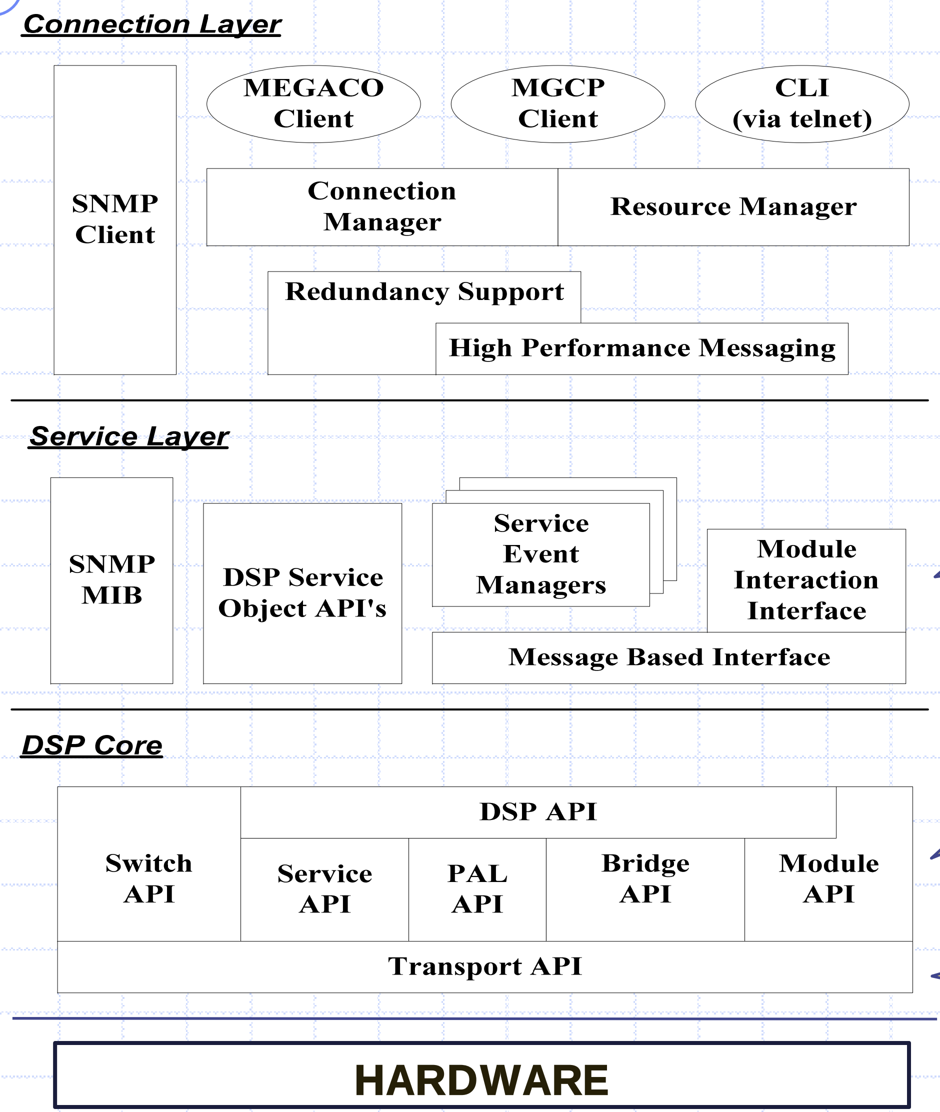
Another example of abstraction

- consider an embedded OS
  - has no user interface component
  - just has an API

- the “target” board contains the embedded processor and a bunch of other resources, in this case
  - switching channels
  - DSPs
    - to implement vocoders, codecs, protocols etc
  - the basic services on the board are accessed by a very basic API that uses a 1:1 mapping between C function calls and processor opcodes

- the OS needs to provide an abstraction
  - what we care about is making phone calls, so how about …

Layers of abstraction in a sample API …



The highest level of abstraction implements and manages the concept of ‘connections’ between phones

This level combines system calls to chips to provide higher level services, such as send/receive of messages

There APIs abstract individual physical chips on the board

Bottom layer just communicates to the board

OS Responsibilities...

The OS has a contract. It has an obligation to the rest of the system. This obligation includes:

- The OS has to make sure that programs ***only use resources when they are available***
- The OS has to make sure that programs ***only access resources that they are allowed to***
- The OS has to has to make sure that many programs can ***share access to system resources*** .... ie ... OS facilitates ***'Multiprocessing'***
- The OS has to be *trusted* ... ie ... the OS has to work exactly as specified. This part of the system is ***mission critical and cannot fail***

A Short History of Operating Systems

Generations:

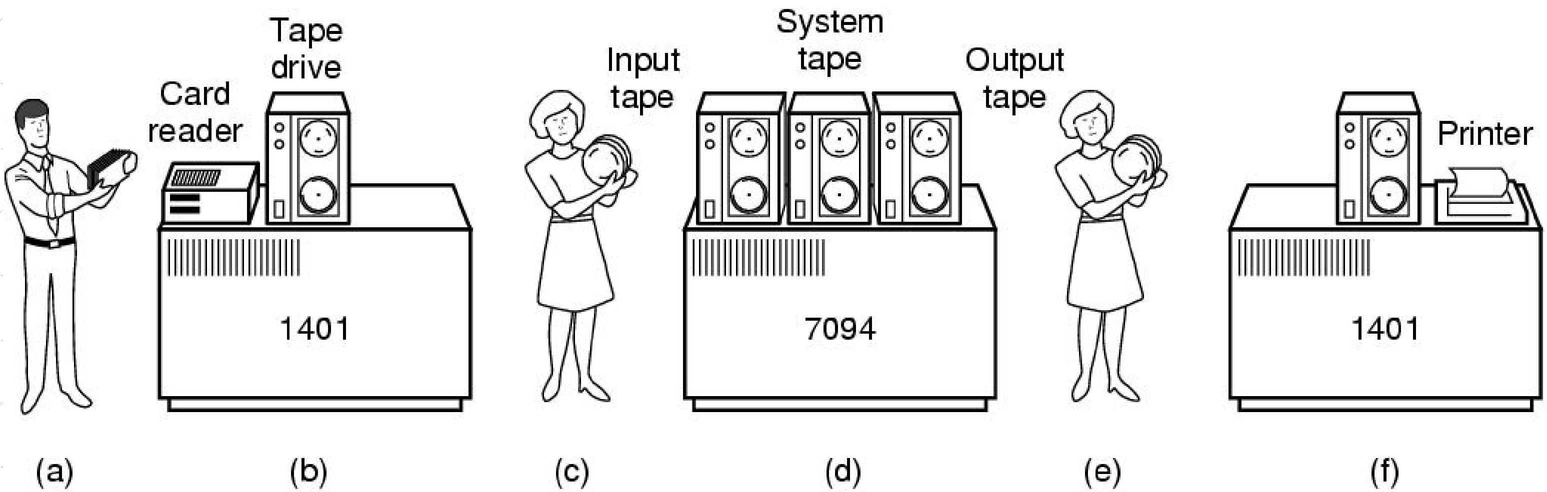
(1945–55)	Vacuum Tubes
(1955–65)	Transistors and Batch Systems
(1965–1980)	ICs and Multiprogramming
(1980–Present)	Personal Computers

Batch Systems

IBM 370 (1970)

- OS processes “jobs”
- “jobs” are non-interactive
- “jobs” are controlled by a sequence of input commands

Key Concept: Batch Processing



(a) Programmers bring cards to 1401.
(b) 401 reads batch of jobs onto tape.
(c) Operator carries input tape to 7094.
(d) 7094 does computing.
(e) Operator carries output tape to 1401.
(f) 1401 prints output.

A Batch Job (Card Stack for a Job)

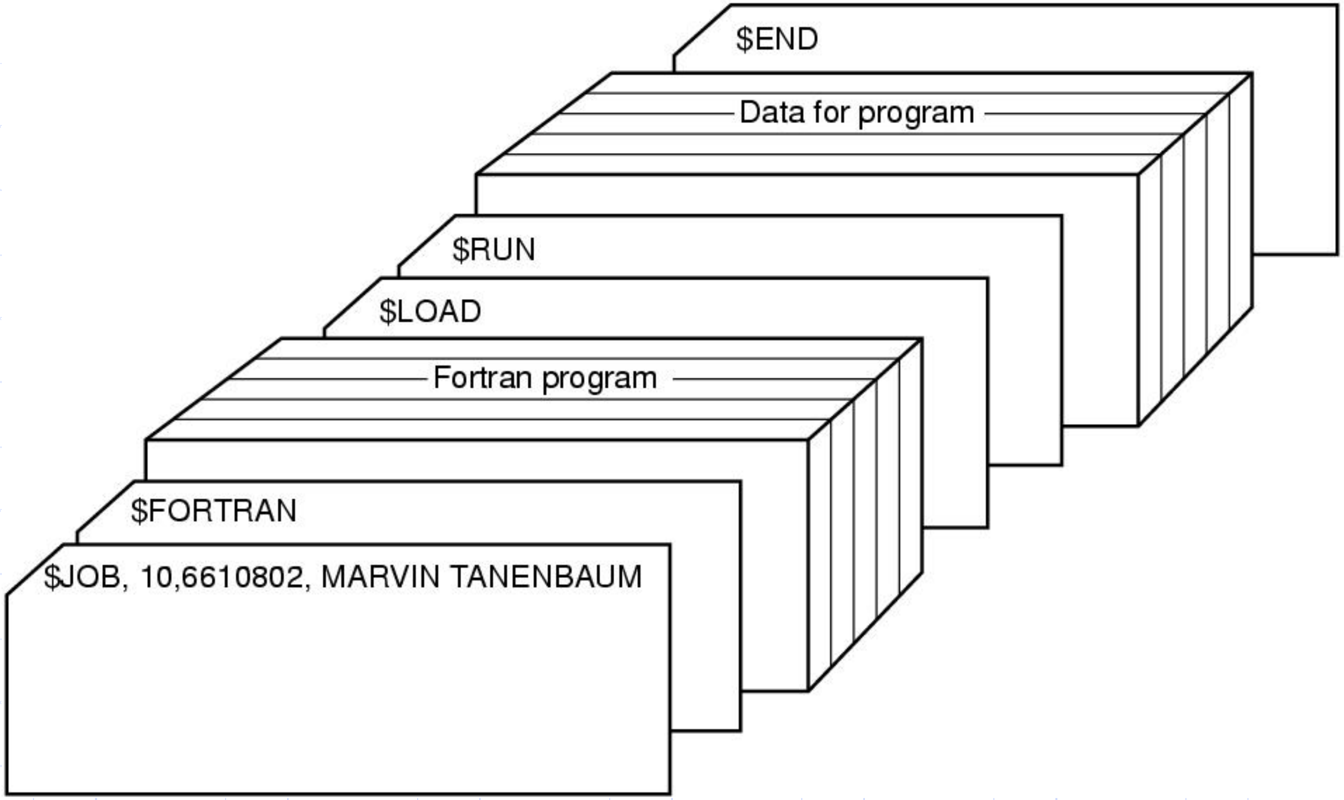


Figure 1-4. Structure of a typical batch job.

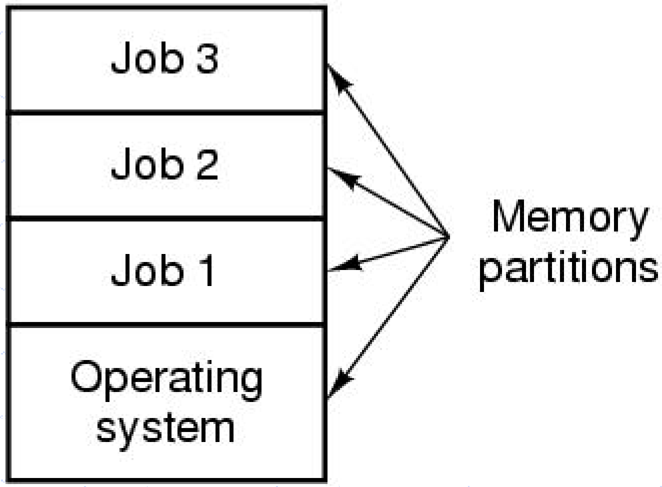
Batch Processing (Sample JCL)

```
//SIMOJOB1 JOB (ACCTINFO),CLASS=A,MSGCLASS=C,NOTIFY=USERID
//*
//TEMPLIB1 DD DISP=(NEW,CATLG),DSN=&DSNAME,
//            STORCLAS=MFI,
//            SPACE=(TRK,(45,15,50)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PO)
//*
//JOBLIB DD DSN=SIMOTIME.DEVL.LOADLIB1,VOL=DCB083
//*
//STEP0100 EXEC PGM=PROGRAM1
//*
//STEP0200 EXEC PGM=PROGRAM2
//*
//STEP0300 EXEC PGM=PROGRAM3
//STEPLIB DD DSN=SIMOTIME.DEVL.TEMPLIB1,DISP=SHR
//*
//STEP0400 EXEC PGM=PROGRAM4
```

Key Concept: Multiprocessing

- ***Multiprocessing*** is the term given to the sharing of a computer and its resources by more than one user, or more than one job
- Operating systems that supported batch processing were the first to incorporate multiprocessing
- The idea was to make efficient use of CPU time when jobs were waiting for I/O devices or for operator intervention

Multiprocessing of Batch Jobs



- Memory is divided into fixed partitions
- Each job is loaded into a partition
- OS runs on job at a time until the job blocks (eg: waiting for a tape to be mounted etc)
- When a job is blocked the CPU will switch to a different partition

Note:

- resources such as tape drives, printers, volumes (storage disks) are allocated to a job and held until the job finishes
- only one job can use a resource at a time

Other Batch Systems

Dennis Richie and Ken Thompson's design of the first version of UNIX on a PDP-7 marked the beginning of a long relationship between DIGITAL and UNIX that continues to the present. In the early 1970s UNIX was ported to a PDP-11 and in 1983, DIGITAL developed ULTRIX, its own version of the popular operating system. Here, Richie and Thompson work at a PDP-11.

Unix **batch** man page

NAME

batch - queue, examine or delete jobs for later execution

SYNOPSIS

batch [**-V**] [**-q** *queue*] [**-f** *file*] [**-mv**] [**TIME**]

DESCRIPTION

batch reads commands from standard input or a specified file which are to be executed at a later time, using the shell set by the user's environment variable **SHELL**, the user's login shell, or ultimately **/bin/sh**.

Key Concept: Timesharing Systems

Timesharing

- Uses multiprogramming to enable multiple ‘simultaneous’ users
- Support interactive computing model (illusion of multiple consoles)
- Different scheduling & memory allocation strategies than batch
- Considerable attention to resource isolation (security & protection)
- The goal was to optimize to optimize response time for users

PCs and Workstations

- Fourth generation Operating Systems are what we have today with XP, Vista, Linux, MacOS etc
- These OSs are designed primarily for single users (although they support multiple users)
- Make use of inexpensive hardware resources
- Initially these were ‘bare bones’ systems (eg: CP/M) to run programs on
- Now they are sophisticated and more complex than any of their predecessors

We will discuss these in more detail when we look at

the Linux and Vista case studies

Summary of Key Concepts

- OS manages the physical resources of the computer
- OS provides resource abstraction to better let us use the computer as a tool (ie: to perform tasks <like gaming> that we want to perform)
- OS makes sure that resources are shared safely and fairly between processes
- OS enables multiprogramming
- OS needs an interface for programmers, or it is useless
- OS typically (but not always) needs a user interface
- Batch systems work in an “offline” mode
- Timesharing systems work in an “online” mode
- Modern OS’s support both batch and timesharing operations

The End