CSCI 418—Operating Systems

Lecture 2

Memory Management–early systems

Textbook: Operating Systems — Internals and Design Principles (9th edition)
by William Stallings

1. A brief history of OS development

- **1. First generation (1940–1955)**, mainly used in military.

No serious OS. Military app.

- **2. Second generation (1955–1965)**, mainly used in business.

Commercial app. COBOL. Computer operator.

$JOB (Binhai)
$PASSWORD: —
$LANGUAGE: FORTRAN → load data ... $EOJ.

CPU is fast
I/O is slow.

Spooling:

- **3. Third generation (1960s–late 1970s).**

multiprogramming: several users share a cpu.

1. interrupts.
2. scheduling

Unix: 20,000 lines.
Ken Thompson

- **4. Post-3rd generation (late 1970s–early 1990s).**

- Virtual memory: the entire program doesn't have to reside in main memory.

App: DBMS, PC, high-speed network, parallel computing.

- Windows 95, 98 (lines: 14M+)

- **5. Modern generation (mid-1990s–now).**

everything is hooked to the internet

2. Types of OS

- **1. Batch system**. Example: Systems processing punched cards, tapes, etc.

- **2. Interactive system**. Example: DOS running on a PC.

- **3. Real-time system**. Example: High speed aircraft, cruise missile.

- **4. Hybrid system**. Example: Combination of batch and interactive system, e.g., CM-5.
- **5. OS for intelligent phone**. Example: Android.

- **6. Embedded system**. Example: Kernel for a robot, e-car.

3. Early Memory Management Systems

- In the early days, a computer can only have one user at one time. Moreover, a computer can only run a program at a time. To run a program, it must be entirely and contiguously loaded into memory. The memory management is therefore easy.
- **Algorithm**: Load a job in a single-user system
  - 1. Store first memory location Y of program into base register
  - 2. Set program counter = Y
  - 3. Read first instruction of program
  - 4. Increment program counter by # of bytes of instruction
  - 5. Last instruction?
  - if YES, then stop loading
  - if NO, then continue with step 6
  - 6. Program counter > memory size?
  - if YES, then stop loading
  - if NO, then continue with step 7
  - 7. Load instruction to memory
  - 8. Read next instruction of program
  - 9. Go to step 4

4. Fixed (Static) Partitions

- Single-user system cannot support **multiprogramming**, which is especially not cost-effective in the business community.
- Static partition is one way to handle multiprogramming.
- Once the system is power on and reconfigured, the partition sizes remain static. Partition sizes can only be changed/reconfigured when computer is rebooted.
- Any program must be entirely and contiguously stored in a partition.
- Clearly, several programs (jobs) can reside in memory at the same time.
- What is the drawback of fixed partition?

internal fragmentation —— partial use of fixed partitions.

▷ partition size is too large — waste memory

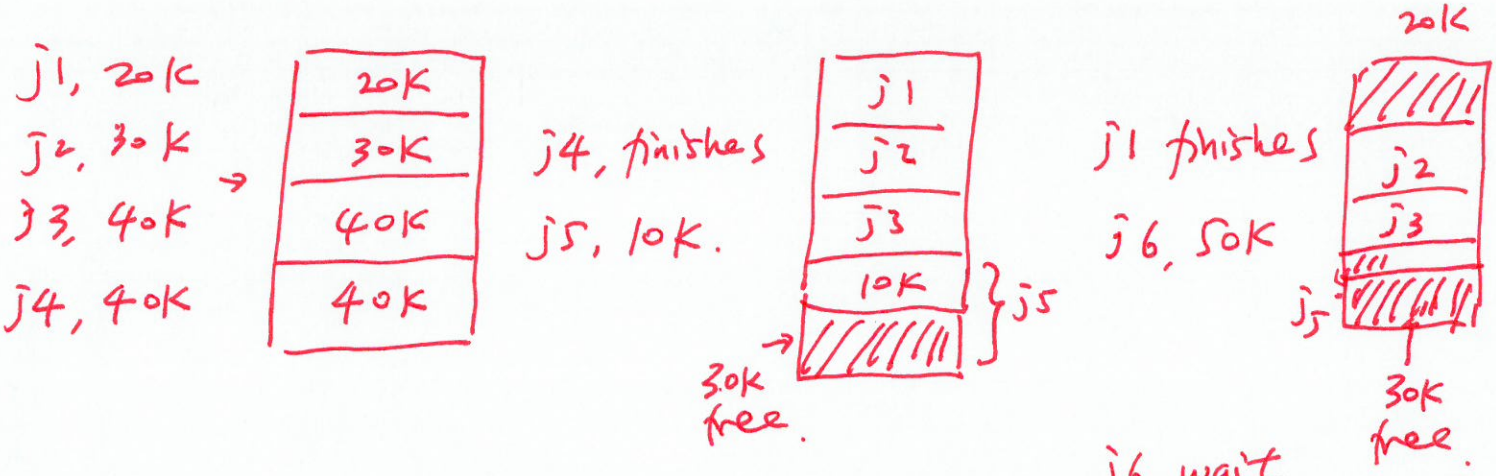
▷ partition size is too small — large jobs will be rejected.

- **Algorithm to load a job in a fixed partition**

```
– 1.  Determine job's requested memory size.
– 2.  If job_size > size of largest partition
–       then reject the job
–          print appropriate message to operator
–          go to step 1 to handle the next job
–       else continue step 3.
– 3.  Set i = 1.  //i is the counter
– 4.  While i <= number of partitions in memory
–       if job_size > memory_partition_size(i)
–          then i = i +1
–       else
–          if memory_partition_status(i)='FREE'
–             then load job into memory_partition(i)
–                  change memory_partition_status(i)
–                  to 'BUSY'
–                  go to step 1
–          else i = i + 1
– 5.  No partition available at this time, put job
–     in waiting queue.
– 6.  Go to step 1 to handle next job.
```

5. Dynamic Partitions

- There is no partition when the computer is turned on.
- Available memory is still kept in contiguous blocks but jobs are given only as much memory as they request.
- It still does not solve the memory-wasting problem completely.



external fragmentation: within dynamic partition, the allocation of memory creates fragments of free memory between blocks of allocated memory which are wasted.