CSS 503A Lecture #8 2019-04-24

Administrivia

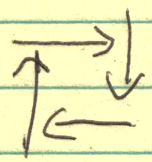
(minor revisions to Asn 2 by end of Friday)

Previously on CSS 503A...

- dining philosophers problem

- sleeping barbers problem
  - assignment 2

- gridlock: (automobile) traffic version of deadlock



- deadlock: mutual dependency

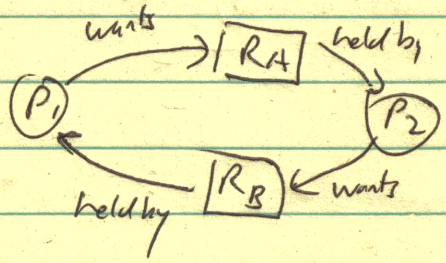
- in practice: OS lets user deal with deadlock ("why is this taking so long?")
  - prevention
  - detection & recovery

- OS may not avoid deadlocks, but your system should - if possible
  - famous approach: strict ordering of resources

Previously (cont.)

- 4 conditions
  1) mutual exclusion (exclusive access to resources)
  2) hold & wait
  3) no pre-emption
  4) circular wait

- 2-color graph: resources & processes



- graph algorithm: find loops (cycles)
  - DFS or BFS

OSTEP (cont.)

- Avoidance: attack one of the 4 conditions
  1) disallow mutual exclusion
  2) disallow hold or disallow wait
  3) allow preemption
  4) prevent circular wait
     - strict ordering of resources

- detection: construct resource allocation graph & look for cycles

- recovery: kill process(es) until no cycle

Banker's Algorithm

- multiple instances of resources (resource class)
  → can't use resource graph

- each process must declare (in advance) Maximum use

- resource requests will be granted only if there is a sequence such that all processes can successfully acquire & release resources

Banker's Algorithm (cont.)

m = # resource classes

n = # processes

Available $[m]$ = # resources available for each class

Max $[n, m]$ = maximum demand for each resource, for each process

Allocation $[n, m]$ = current allocation of each resource, for each process

Need $[n, m]$ = Max − Allocation

Each row is a vector with one element for each resource class

$$\vec{x} < \vec{y} \text{ iff } \bigwedge_{i=0..m-1} x_i < y_i$$

Banker's Algorithm (cont.)

- safety algorithm (lemma? subroutine?)

```
Work [m]
Finish [N]

Work = Available
Finish = False
loop:
    find i s.t. !Finish[i] and Need[i] < Work
    if no such i:
        system is safe iff Finish == True
        return
    Work += Allocation [i]
    Finish[i] = true
```

Work, Available: currently available resources (some processes may be holding resources - when they terminate, their held resources become available)

i: process i such that its maximum request could be satisfied

Assume process i gets all its maximum requested resource and is allowed to run to completion, then its currently-held resources become available

- maintain the state such that, at any point in time, there exists a potential allocation sequence that doesn't result in a deadlock

Banker's Algorithm (cont.)

Allocation:

$\vec{Request}[i]$: request vector of process i

if $\vec{Request}[i] > \vec{Need}[i]$

    raise exception

if $\vec{request}[i] > \vec{Available}[i]$

    wait

if system would be unsafe after allocation

    wait

grant requested resources

Banker's Algorithm (cont.)

example: resources: A = 10, B = 5, C = 7

current Available = (3, 3, 2)

	Max	Current Allocation	Need = max - current
P_0	(7,5,3)	(0,1,0)	(7,4,3)
P_1	(3,2,2)	(2,0,0)	(1,2,2)
P_2	(9,0,2)	(3,0,2)	(6,0,0)
P_3	(2,2,2)	(2,1,1)	(0,1,1)
P_4	(4,3,3)	(0,0,2)	(4,3,1)

$\langle P_1, P_3, P_4, P_0, P_2 \rangle$ → satisfies safety

- P_1 can be satisfied immediately
- P_0, P_4 would have to wait

File System - Interface

- character-at-a-time
  - keyboard
  - tty
  - speaker
- block-at-a-time (record-oriented)
  - punch card
  - line/page printer
  - disk (sectors)

- sequential
  - tape
  - tty
  - punch card
- random-access
  - disk
  - flush
  - bit-map screen

Pipe - pseudo device

File System - Interface (cont.)

- input (read) only
  - card reader
  - keyboard
- output (write) only
  - printer / card punch
  - bit-map screen
- input / output (read/write)
  - disk
  - tape
  - flash

File System - Interface (cont.)

- disk drive: mechanical device
  - electric impulse: stepper motor, read/write head
  - controller addresses numbered blocks

- working with raw devices: inconvenient
  - files: higher level of abstraction

- File abstraction:
  - non-volatile storage (* except tmpfs)
  - logically contiguous space
  - attributes:
    - name
    - type
    - size
    - protection/permissions
    - time/date (create/access/modified)
    - ownership (user, workgroup)

- disk: shared by multiple users
  → File permissions & ownership

File System Interface (cont.)

- File location:
  - user view: directory
  - kernel/physical view: disk blocks

- format
  - sequence of bytes
  - fixed-size record (80 bytes, 512 bytes, ...)
  - variable-size records (ISAM / VSAM)
    - → B-Tree
  - text
  - binary
  - application-specific format
    - Word doc vs ELF vs gzip
    - with or without OS support

- file type: "extension"
  - meaningful on some systems
  - ignored by BSD/Unix/Linux systems

- "Magic Number" - fast identifier check on file content
  - application-specific — maybe file type vs. definitely not

- Unix "file" command: heuristic

File Operations

- open
  - open existing file
  - create file if it doesn't exist
  - only create new file
    → atomic operation
      - may be used for synchronization tools with NFS
  - open for reading / writing / both
    - set file pointer to beginning
    - set file pointer to end (append)
- read
  - read specified number of bytes
  - advance file pointer
- write
  - write # bytes to disk & advance file pointer
- close
  - flush cached data to disk
  - release resources (buffers, handles)

- tell()
  - return current file pointer position
- seek
  - reposition file pointer
- ioctl()
  - device control function
- fcntl()
  - file parameters

Unix-like

```
int fd = { open ( fname, O_RDONLY)
         { open ( fname, O_WRONLY,
                          S_IRUSR / S_IWUSR)
read ( fd, buf, sizeof (buf))
write ( fd, buf, sizeof (buf))
close (fd)
```