Exam Notes of Code and Math

Wednesday, December 6, 2023 5:43 PM

Amdahl's Law

$$\text{speedup} \leq \frac{1}{S + \frac{1-S}{N}}$$

S: % of series
N: Number of processing cores

Banker's Algorithm

System Snapshot

Need = Max − Allocation

	Allocation	Max	Available	Need
	A B C D	A B C D	A B C D	A B C D
T_0			Only 1 entry	
T_1				
T_2				
T_3				
T_4				

Safe State:

Follow Algorithm:

1) Check if need has each element less than or equal to available.

2) If satisfies, then add thread to safe state and add need to available

3) If does not satisfy, go to next one.

4) If no more threads, then safe state exists.

5) If after full cycle no more threads can be added but still some exist not in the safe state, then false.

e.g. <T0, T2, T3, T4, T1>

Request for Immediate Access

- If the elements in Available ≥ Request, access is granted IMMEDIATELY

e.g. T_1 request for <1 4 10> is granted immediately

Resource-Allocation Tables

- Determine if something has deadlock
- T_n: Threads R_m: Resources

Traits of dead lock
- Cycle occurs (directed cycle) (Most of the time)

- If does not deadlock, then list order of threads
- Otherwise, list cycles

CPU Scheduling

Given a time quantum, overhead from context switching, and tasks

calculate CPU utilization

Memory Management

Given n-bit address, p-bit physical address, and page size

1) Find # of entries in conventional single-level table

i) Determine # of bits in page size
e.g. if page size = 8KB, then convert to 2^{13} bytes, thus 13 bits

ii) Subtract # of bits found in i with n-bits to get q-bits

iii) $2^{q\text{-bits}}$

2) Find # of entries in an inverted page table.

i) Use # of bits in page size

ii) Subtract from # of bits in physical address to get q-bits

iii) $2^{q\text{-bits}}$

3) Max amount of physical memory = $2^{\text{physical bits}}$

Demand-Paged Memory

Variables: m: Memory Access Time
r_n: rate for a page fault event to happen
S_n: time it takes for such event to complete

Calculate: P: Page-fault Rate

$$\text{Effective Access Time} = (1-P)m + P \cdot \sum_{i=0}^{n} r_i \cdot S_i$$

$$= (1-P)m + r_0 S_0 P + r_1 S_1 P \ldots$$

Virtual Memory

Page Fault Algorithms:

LAU: Least recently used

FIFO: First in first out

Optimal Replacement: Replace the one that has the next value furthest away or does not exist in the rest of the lists.

File Management

Given pointers, size of disk sectors, find max file size. (Size of int = 4)

Type of pointers: Direct, Single, Double, Triple, ...

How to calculate:

$$\left(\frac{\text{size of disk sector}}{\text{size of int}}\right)^{\text{Degree of pointer}} \cdot \text{size of disk size}$$

Sum up all the pointers!

Parent/Child Functions

fork(): Creates duplicate function (1 child 1 parent)

When called as follows:

```
pid_t pid;
pid = fork();
```

If pid < 0, fork failed
pid == 0, child process
pid > 0, parent process

Parent & Child execute every line after fork!

Each fork doubles # of processes

wait(NULL): Used on parent function to wait for children functions to terminate

If not used, one of 2 functions could occur:

Zombie Function:
- Process that has finished execution but still has entry in process table
- Child process done, but parent function is waiting (e.g. sleep(50): sleep for 50 seconds)
- Wastes system resources
- Child finishes first but waits while alive

Orphan:
- Process where parent no longer exists (terminated or finished) without waiting for child to finish
- Orphan process stays and can not properly terminate

Functions to Remember

exec functions execute functions

```
execvp(args[0], args);
```
- Executes and creates new child function
- Child process does not have to run same program as parent process

args[0] & args are pointers to char *args[].
- execvp parameters are ALL pointers
- First variable is a filename

```
execv(args[0], args);
```
- Same as execvp BUT args[0] is path to executable

Threads

- Lightweight process
- Sequential execution stream
- Process has at least 1 thread

Most apps today are multithreaded
- Displaying graphics
- Keystrokes
- Spelling & grammar

- ↑ efficiency, simplify code
- Parallel coding

Thread Creation

0) Import pthread.h
```
#include <pthread.h>
```
1) Declare thread ID
```
pthread_t thread_id;
```
2) Create thread with pthread_create():
```
pthread_create(a, b, c, d);
```
where
- a: pointer to thread ID (&thread_id)
- b: Attributes. Usually NULL or 0 for default
- c: function
- d: function variables

3) Use pthread_join(); for threads (same as wait())
```
pthread_join(a, b)
```
where
- a: thread ID (&thread_id)
- b: NULL

Thread Termination

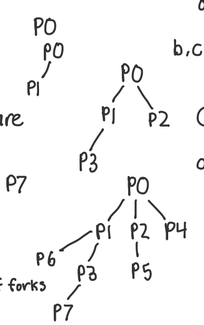
1) Use pthread_cancel(tid);
- Only indicates a request to terminate thread
- Actual termination depends on target thread set up to handle request
- Can set cancellation state & type using API
- Only occurs when pthread_testcancel() invoked, then cleanup handler

Tree Hierarchy

- Start with P0 (parent)
- First fork: P1 is created
- Second fork: P2 and P3 are created
- Third fork: P4, P5, P6, and P7 are created

Continues for each fork

- # of processes = $2^{\text{number of forks}}$



Gantt Chart

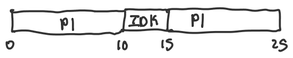
- Chart that displays time for each process
- Shows dependency relations

- Suppose $T_1 = 24$ s, $T_2 = 5$ s, $T_3 = 6$ s

Turnaround Time: Time end − Arrival Time
- Important for something like a round robin Scheduler

Arrival Time ≠ Start Time!

Wait Time: Sum of idle time not running the process when process has started



IDK is the wait time

Include Start Time − Arrival Time

CPU Utilization Rate: 100% $\frac{\text{Total Time} - \text{Total Wait}}{\text{Total Time}}$

Exponential Average Formula

$$T_{n+1} = aT + (1-a)T_0$$

Given a and T_0

- If a = 0, then formula has consistent solution (T_0)
- If a ≠ 0, then in terms of T, which means most recent behaviour is much higher weight than past history.

Producer Consumer Program

- If program has variables (a, b, c)
- Suppose a is an int, b & c are binary semaphores

a: Increments iff a++ or a-- or integer operations

b, c: Wait makes it go from 1→0, Signal makes it go from 0→1

Critical Sections: Operations on a (a++ or a--)

a: Represents if consumer consumed. Depending on code and implementation, CAN BE NEGATIVE (but bad)