Final Review

Glenn Bruns
CSUMB

Structure of the final

- Fully online
  - make sure you can get good internet service
  - for backup, figure out how to use your phone as WiFi hot spot
  - during the exam, let me know if you have a significant connectivity problem
- Two parts
  - part 1: short questions
  - part 2: longer questions
- Open books, open notes, "closed neighbor"

Preparing for the final

- Keep major learning outcomes in mind!
- Make flash cards and test yourself
- Make notes that you can refer to easily during exam
  - flash cards
  - where to find things in book and lecture slides
- Avoid skimming through slides except to take note of things you need to work on
  - skimming through slides is not effective for learning

Process Management Review

Process Management

- Main goal: virtualize the CPU
- Make is so programs are not aware that CPU is being shared
- Main technical problem: how to achieve multiprogramming

OS data structure for processes

The OS manages processes

For each process there is a data structure containing things like:

- the process id
- the process state
- the process register values
- the size of process memory

In Linux it's the `task_struct` data structure

How to support multiprogramming

- how to avoid excess overhead?
  - let processes run directly on the CPU
- how to keep limit processes control over hardware?
  - hardware: user/kernel mode of CPU
  - hardware: traps for system calls
- how to stop a running process?
  - hardware: hardware interrupts

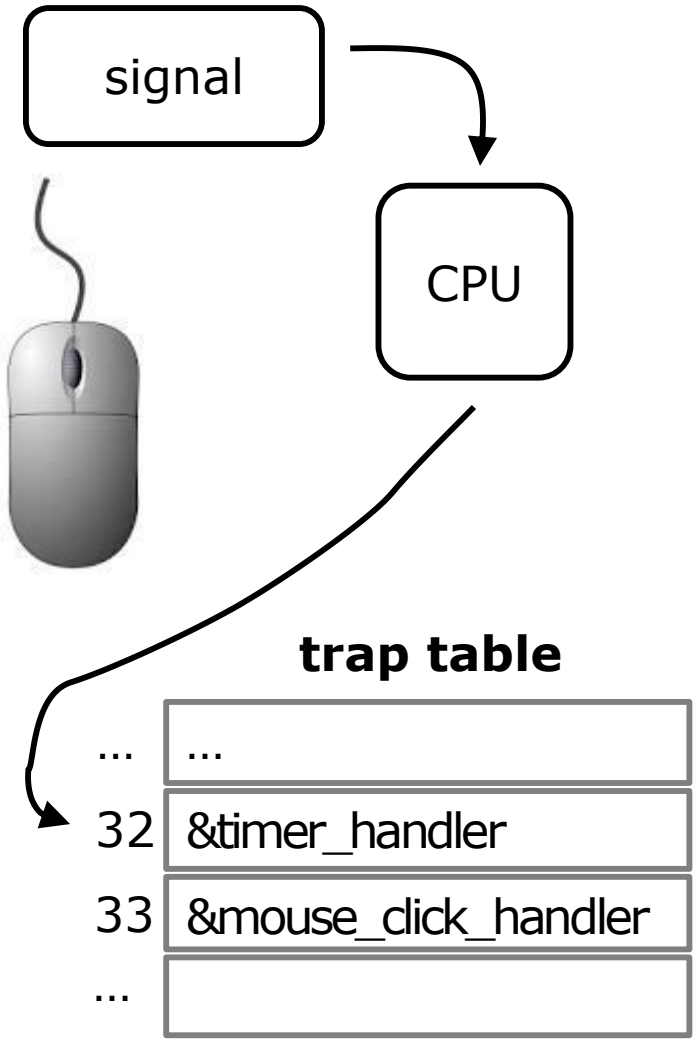
How to process hardware interrupts?

When certain hardware events occur, the CPU:

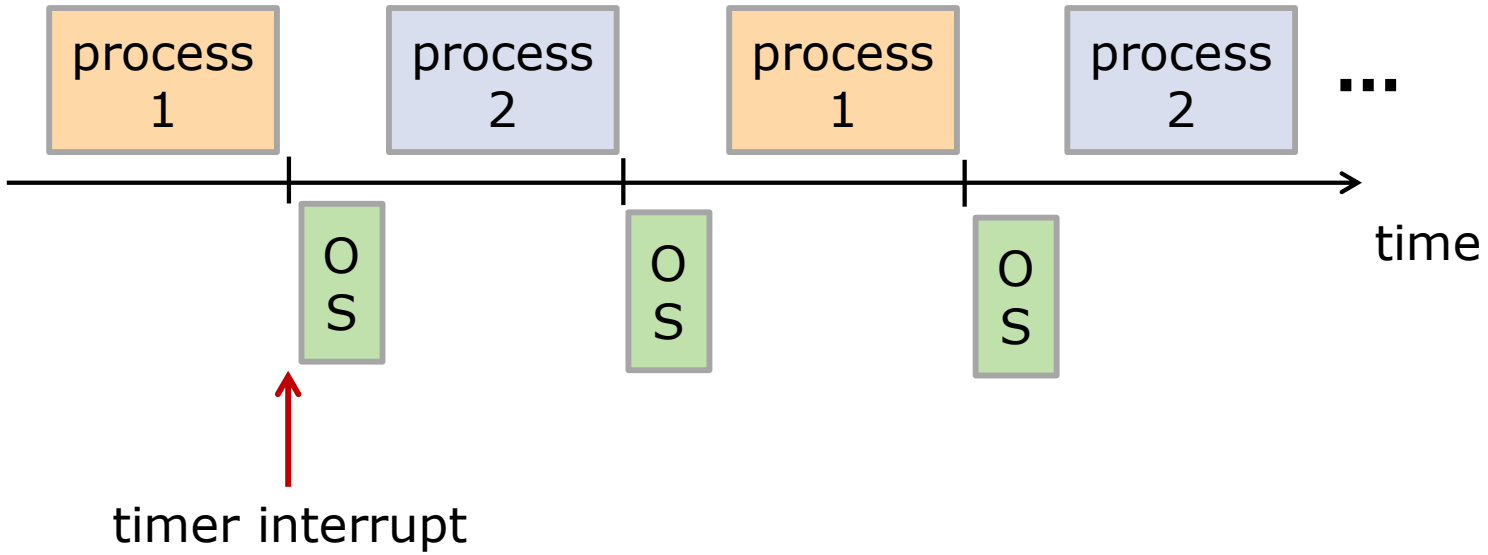
- stops the current process, saves its registers, enters kernel mode
- locates an interrupt handler in the OS trap table; the OS runs it

When the handler finishes, it performs a special instruction, then the CPU:

- restores the process register values, enters user mode, and restarts the process



Context switching with timer interrupts



1. hardware (CPU):
   - save P1 register values
   - switch to kernel mode
   - jump to interrupt handler

2. OS:
   - save P1 register values to P1 proc-struct
   - restore P2 register values from P2 proc-struct
   - RETURN-FROM-INTR

3. hardware:
   - restore P2 register values
   - switch to user mode
   - restart process 2

Scheduling algorithms

Shortest-job first

First-come, first-served

Shortest time to completion first

Round robin

Multi-level feedback queue

Scheduling criteria

How do we choose among possible process scheduling policies?

- turnaround time
- response time

Question

Which scheduling policy is preferred for low average response time?

a) shortest job first

b) first-come, first-served

c) round robin

Answer

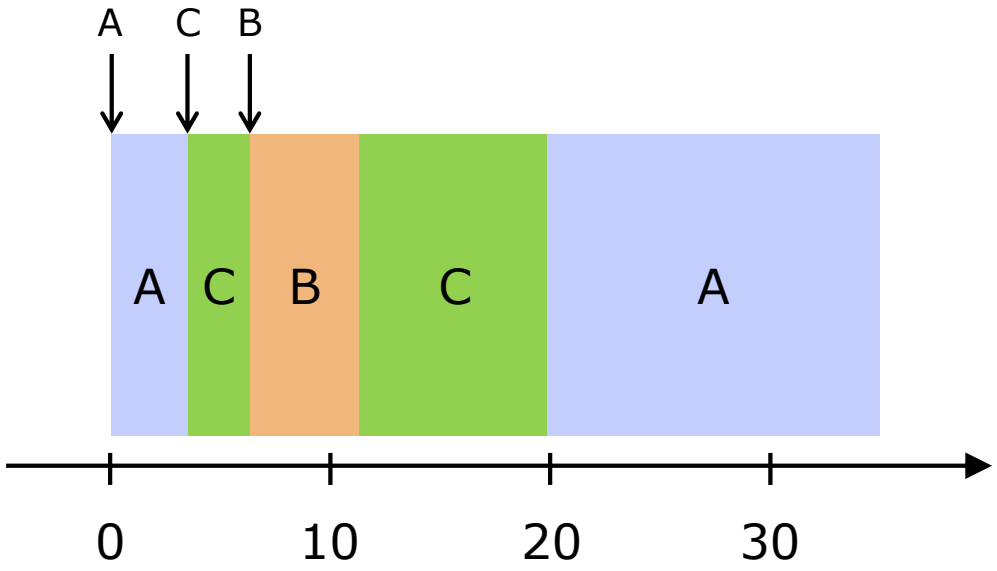
Which scheduling policy if preferred for low average response time?

a) shortest job first

b) first-come, first-served

c) round robin

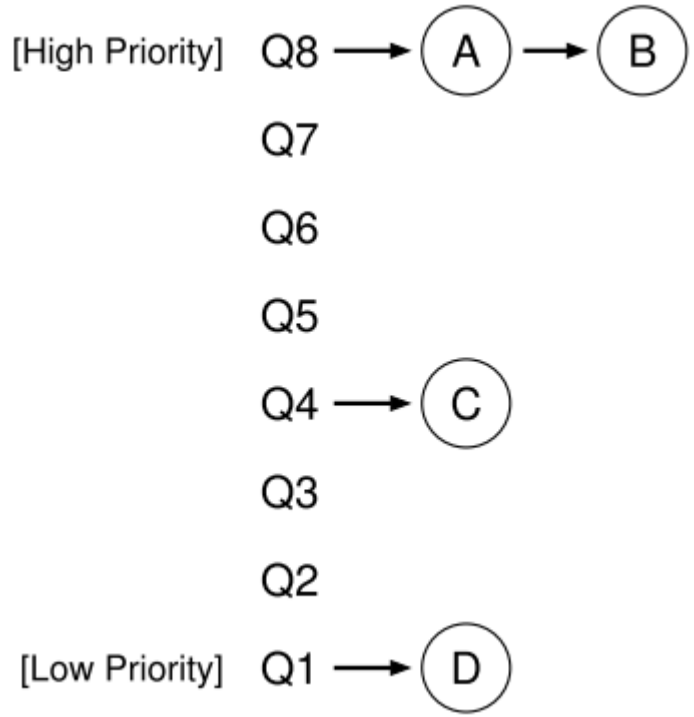
Shortest time to completion first (STCF)



Whenever a job arrives, run the job that would finish first.

Here, C “preempts” A, and then B preempts C.

MLFQ main idea: use job queues



1. Put jobs on queues according to priority
2. Run jobs on higher-priority queues first
3. Run jobs on same priority queue using round robin

(diagram from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

MLFQ rules for updating priorities

condition	change to priority
job starts	highest priority
job uses up time allotment at a given level	reduce priority level by 1
end of time period S	set all jobs to highest priority (boosting)

Memory Management Review

Memory Management

- Main goal: virtualize memory
- Each process gets its own virtual address space
- Main technical problems:
  - how to do address translation quickly?
  - how to avoid using too much space for page tables?

Question

Suppose each process's virtual address space were mapped to a single, contiguous physical range of addresses. What's the problem?

a) slow translation

b) wasted space

c) no problem

Answer

wasted space

Normally, lots of the virtual address space is not used.

In particular, the area between the stack and heap.

Question

Segmented virtual memory leads to memory fragmentation. Would making all segments the same size fix the problem?

a) yes

b) no

c) partly

Answer

partly

External fragmentation would be reduced, but:

- there might be wasted space within the same-sized pieces of memory

Question

In virtual memory with paging, are the physical and virtual page sizes the same?

Answer

Yes – always!

Question

What is the physical counterpart to a “Virtual page number”?

Question

Physical Frame Number (PFN)

Question

In simple paging, what are the parts of a virtual memory address?

Answer

there are two:

- virtual page number
- offset (into page)

Question

You have 16 things and need to give each one a unique ID. How many bits do you need for the ID?

Answer

4

With four bits you can create 16 ID values:

0000 (0)

0001

0010

…

1111 (15)

Question

In simple paging with a page size of 4K bytes, how many bits are needed in the offset part of a virtual address?

Hint: 2^10 = 1024

Answer

12

The offset has to be able to identify any of the 4K bytes (4096 bytes). With 12 bits you count from 0 to 4095.

The calculation is:

log2(4096) = 12

Question

In virtual memory with paging, where is the page table stored?

a) in CPU registers

b) in the MMU

c) in memory

Answer

in memory

Why?

Question

If we have a 32 bit virtual address space, and 4KB-sized pages, what is the size of a page table (in bytes)?

(assume simple paging, 4 bytes per page table entry)

Answer

12 bits needed for offset

so 20 bits for virtual page number

so 2^20 pages

If 4 bytes per page table entry, then 4 * 2^20 bytes for the page table

= 4.2 MB (2^20 = 1048576 = 1024 * 1024)

Question

In virtual memory with paging, how many page tables are used?

a) one per process

b) one for the OS

c) depends on the OS

Answer

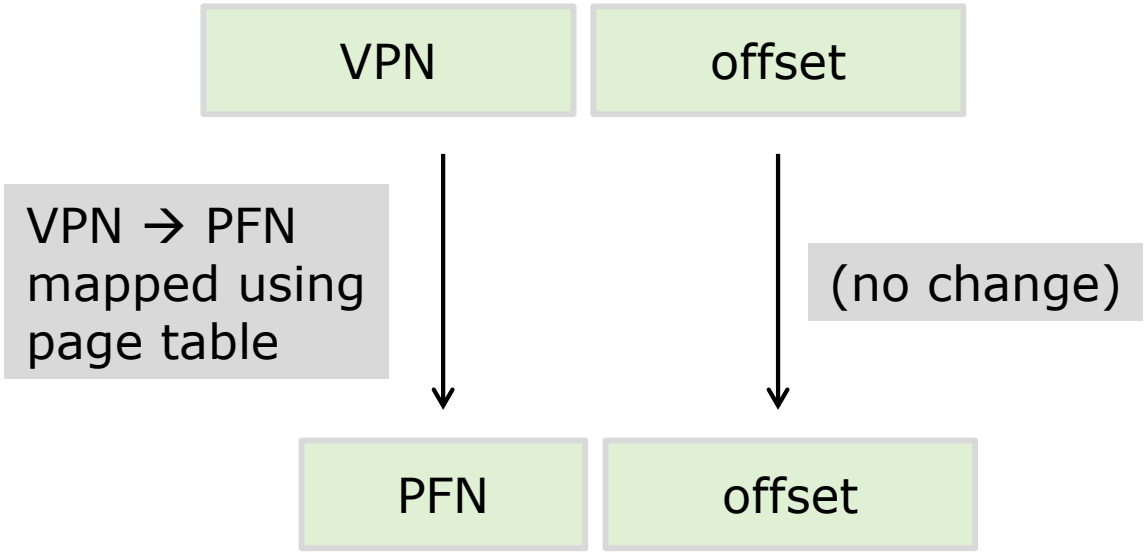
one per process

Question

How is address translation performed with simple paging?

(write or draw a brief explanation)

Answer



# of bits in offset part always the same
PFN can have and VPN can have different # of bits

Question

Where is the TLB?

a) in the MMU

b) in main memory

c) on disk (are you kidding me?)

Answer

a)

In fact, the TLB is usually associative memory

Question

What is the AMAT for a TLB with miss rate of 0.5%, hit time of 1 clock cycle, and miss penalty of 50 clock cycles? (answer in clock cycles)

(also, what does AMAT stand for?)

Answer

.995(1) + 0.005(50) = 1.25 clock cycles

AMAT = “Average Memory Access Time”

Fun Question

Guess typical TLB specs:

a) size (in # of entries)

b) time to process a hit (in clock cycles)

c) penalty for a miss (in clock cycles)

d) miss rate (percent)

Answer

size: 12-4096 entries

hit time: .5-1 clock cycle

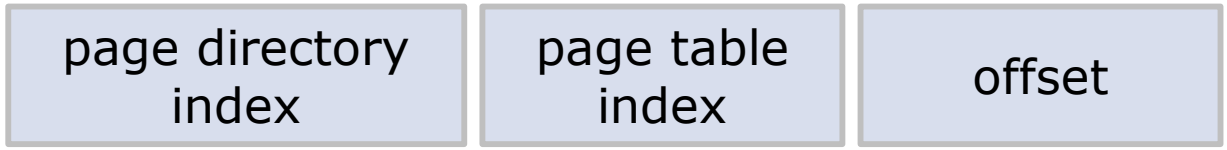
miss penalty: 10-100 clock cycles

miss rate: 0.01 – 1 %

Question

In multi-level paging, what are the parts of a virtual memory address?

Answer

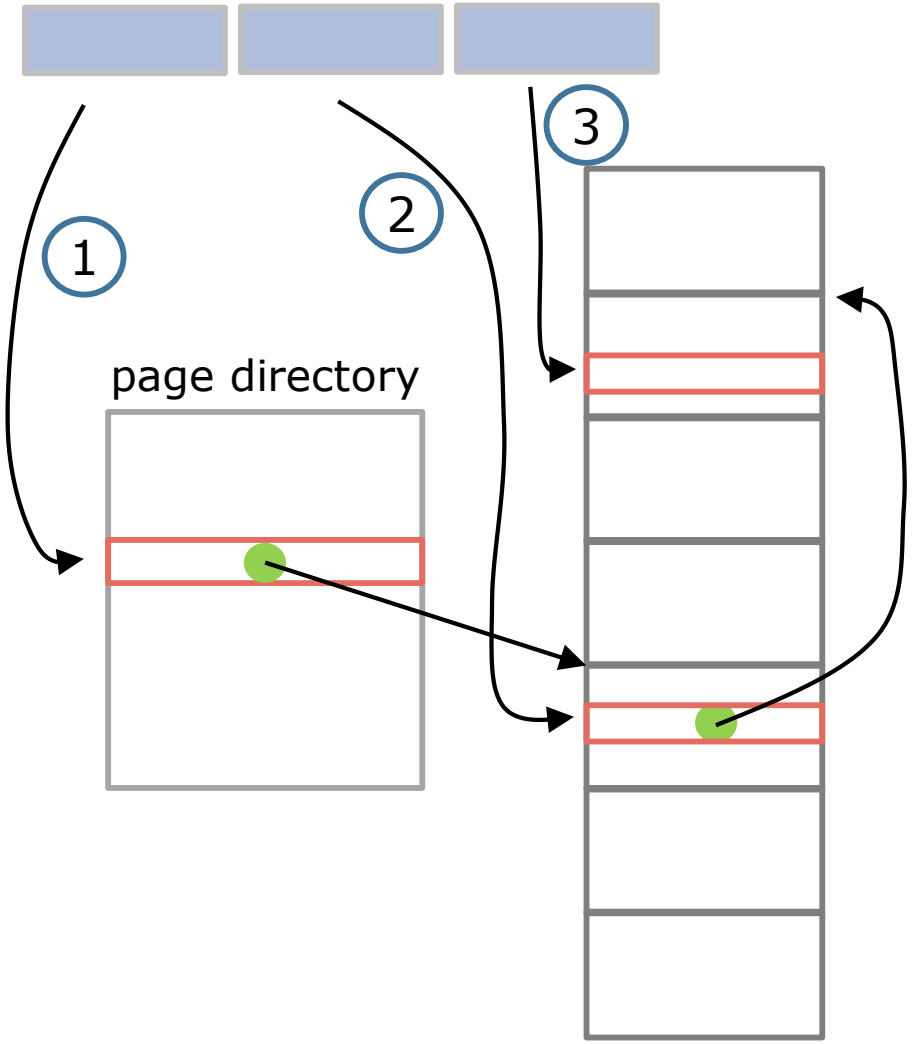


Question

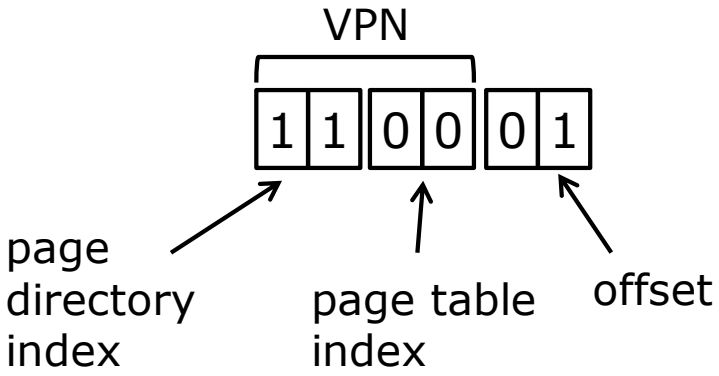
Write down the address translation steps when using multi-level paging.

Answer

1. Use the page directory index to get a get an entry of the page directory. From the entry, get a PFN.
2. The PFN identifies a physical page that contains a "chunk" of page table. Use the page table index to get an entry of this page table chunk. From the entry, get a PFN.
3. Now you have the PFN, so just concatenate it with the offset to get the physical address.



Address translation



page directory

00	1	201
01	0	-
10	0	-
11	1	204

page table

00	12
01	13
10	-
11	100

PFN 201

00	-
01	-
10	88
11	15

PFN 204

1. Get page directory base address from register
2. Add page directory index to get page directory entry; get PFN of page table chunk
3. If valid, add page table index to PFN of page table; get PFN of virtual page

Concurrency Review

Concurrency

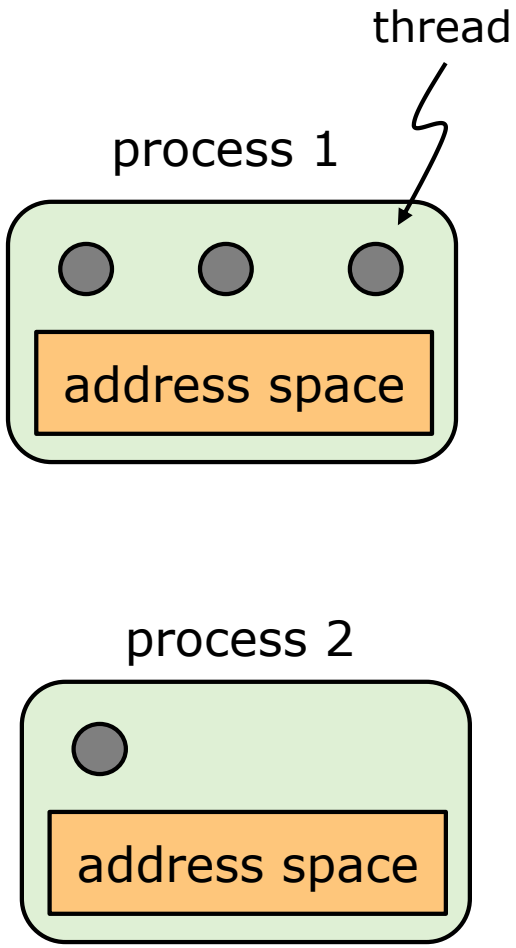
- Main goal: coordinate the execution of processes
  - mutually exclusive access to critical sections
  - avoid deadlocks
  - processes wait efficiently
- Synchronization primitives
  - locks, condition variables, semaphores
  - some operations are blocking operations

What we covered

- concurrency – basic issues
- synchronization primitives (lock, semaphore, cv)
- implementing locks
- pthreads interface
- coding thread-safe objects
- Dahlin methodology for concurrent programming
- code: bounded buffer, rwlock
- semaphores

Threads vs. processes

- Threads and processes both have state
- Threads are faster to create, destroy, and context switch
- The threads within a process share a virtual address space
- But, each thread has its own stack, and its own registers



Question

Fill in the blank:

A _____ is a piece of code that accesses a shared resource, such as a shared variable.

Answer

critical section

You avoid race conditions by preventing more than one thread from reading or writing shared data at the same time

Question

Fill in the blank:

A race condition is when the result of a multi-threaded program depends on

_____

Answer

the timing of their execution

or: their relative speed of execution

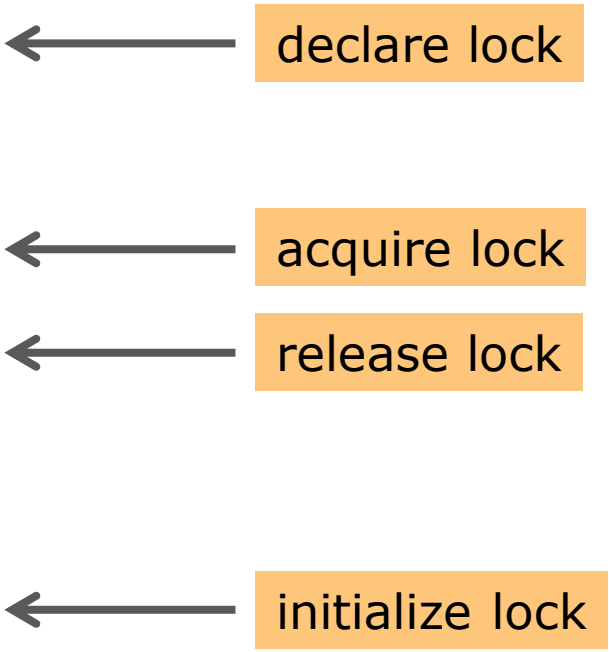
Counting code with locks

```
lock_t lock1;

void *mythread(void *arg) {
    int i;
    for (i = 0; i < 1e7; i++) {
        lock(&lock1);
        counter = counter + 1;
        unlock(&lock1);
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    init(&lock1);
    pthread_t p1, p2;
    Pthread_create(&p1, NULL, mythread, "A");
    Pthread_create(&p2, NULL, mythread, "B");

    // join waits for the threads to finish
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("main: done with both (counter = %d)\n", counter);
    return 0;
}
```



Question

What does it mean for a lock to be fair?

Answer

If a thread requests the lock, it will eventually get it.

Question

```
typedef struct __counter_t {
  int value;
  pthread_mutex_t lock;
} counter_t;

void init(counter_t *c) {
  c->value = 0;
  Pthread_mutex_init(&c->lock, NULL);
}

void increment(counter_t *c) {
  c->value++;
}

void decrement(counter_t *c) {
  c->value--;
}

int get(counter_t *c) {
  int rc = c->value;
  return rc;
}
```

```
Pthread_mutex_lock(&c->lock);
Pthread_mutex_unlock(&c->lock);
```

You are building a thread-safe counter.

Where would you insert lock and unlock operations in this code?

a) at the beginning and end of increment() and decrement()

b) at the beginning and end of get()

c) both a and b

Answer

c)

Question

When you add locks to your code, you should expect:

a) better performance

b) worse performance

Answer

b) worse performance

Threads will sometimes be blocked waiting to acquire the lock.

Question

A signal operation on a condition variable:

a) takes a single thread off the condition variable's waiting list

b) takes all threads off the condition variable's waiting list

Answer

a) single thread.

The broadcast operation takes all threads off a condition variable's waiting list

Question

What happens if signal() on a condition variable that has no threads in its waiting list?

a) signal() has no effect

b) signal() waits until a thread joins the waiting list

c) signal() returns right away, but then the next thread that calls wait() does not have to wait.

Answer

a) signal() has no effect

Note how this is different from what happens if you signal a semaphore that has no threads on its waiting list.

Dahlin rules for conc. programming

1. Always use monitors (locks + cond. vars). Don't use semaphores.
2. Always hold lock when operating on a condition variable
3. Always acquire lock at the beginning of procedure and release it at the end
4. Always use:

```
while(predicateOnStateVariables(...) == true/false){
    condition->wait()
}
```

not an if statement.

"Dahlin method" = "Anderson/Dahlin method"

Simple bounded buffer: class design

```
typedef struct {
  // state variables
  int cnt;      // 0 or 1, depending on whether buffer empty or not
  int val;      // value of item in buffer
} SBUF;

// create a new synchronized buffer
SBUF *sbuf_create();

// write to the buffer
void sbuf_write(SBUF *sbuf, int a);

// read from the buffer
int sbuf_read(SBUF *sbuf);
```

This illustrates how to do OO-style code (without inheritance) in C

Add lock, and code to use it

```
typedef struct {
  // state variables
  int cnt;
  int val;
  // sync. variables
  pthread_mutex_t lock;
} SBUF;
```

```
void sbuf_write(SBUF *sbuf, int a) {
  pthread_mutex_lock(&sbuf->lock);
  sbuf->val = a;
  sbuf->cnt = 1;
  pthread_mutex_unlock(&sbuf->lock);
}

int sbuf_read(SBUF *sbuf) {
  pthread_mutex_lock(&sbuf->lock);
  int a = sbuf->val;
  sbuf->cnt = 0;
  pthread_mutex_unlock(&sbuf->lock);
  return(a);
}
```

- normally a shared object will have exactly one lock
- each method begins and end with locking/unlocking

Add condition variables

```
typedef struct {
  // state variables
  int cnt;
  int val;
  // sync. variables
  pthread_mutex_t lock;
  pthread_cond_t read_go;
  pthread_cont_t write_go;
} SBUF;
```

- if method never need to wait, not condition vars needed
- in this step the designer has a lot of freedom

Question

Fill in the blank

```
int sbuf_read(SBUF *sbuf) {
  pthread_mutex_lock(&sbuf->lock);

  while (                ) {
    pthread_cond_wait(&sbuf->read_go, &sbuf->lock);
  }
  int a = sbuf->val;
  sbuf->cnt = 0;

  pthread_mutex_unlock(&sbuf->lock);
  return(a);
}
```

Answer

```
int sbuf_read(SBUF *sbuf) {
  pthread_mutex_lock(&sbuf->lock);

  while (sbuf->cnt == 0) {
    pthread_cond_wait(&sbuf->read_go, &sbuf->lock);
  }
  int a = sbuf->val;
  sbuf->cnt = 0;

  pthread_mutex_unlock(&sbuf->lock);
  return(a);
}
```

Semaphores (from Downey)

A semaphore is like a variable, but with three differences:

1. when you create a semaphore, you can initialize it to any value, but afterward there are only two operations you can perform
   - increment
   - decrement
2. when a thread decrements the semaphore, if the result is negative, the thread blocks itself and can't continue until another thread increments the semaphore
3. when a thread increments the semaphore, if there are threads waiting, one of them gets unblocks

Semaphores were invented by Edsgar Dijkstra

Semaphore syntax

```
sem = Semaphore(1)
sem.signal()   // i.e. increment
sem.wait()     // i.e. decrement
```

Points to remember:

- before a wait, we don't know if a thread will block
- after a signal, both threads will run concurrently – no way to know which will run immediately
- when a signal occurs, no threads might be waiting

Question

What are all the legal executions of the following code?
Assume semaphore ‘sem’ has initial value 0.

thread A

```
1   statement a1
2   sem.signal()
```

thread B

```
1   sem.wait()
2   statement b1
```

Answer

thread A

```
1   statement a1

2   sem.signal()
```

thread B

```
1   sem.wait()

2   statement b1
```

thread A

```
1   statement a1
2   sem.signal()
```

thread B

```
1   sem.wait()

2   statement b1
```

thread A

```
1   statement a1
2   sem.signal()
```

thread B

```
1   sem.wait()
2   statement b1
```

Question

Extend this code using semaphore(s) to ensure mutually exclusive access to shared variable count.

Don’t forget to show initial values of your semaphore(s).

thread A

```
count = count + 1
```

thread B

```
count = count + 1
```

Answer

```
sem = Semaphore(1)
```

thread A

```
sem.wait()
   // critical section
   count = count + 1
sem.signal()
```

thread B

```
sem.wait()
   // critical section
   count = count + 1
sem.signal()
```

A “symmetric” solution

Language Processing Review

Language processing

- Main goal:
  - define the syntax of a language
  - write code to check for legal syntax (parser)
  - extend parsing code so that it builds syntax tree
- Techniques used:
  - use BNF to define syntax
  - rules to transform BNF definitions
  - predictive parsing

Question

Which of these grammars produces one or more 'a' tokens, separated by commas? (A is the start symbol)

a)

```
A ::= a , A  | ""
```

b)

```
A ::= a B
B ::= , a B | ""
```

Answer

b

a) A ::= a, A | ""

can derive "" "a," "a,a," etc.

b) A ::= a B
B ::= , a B | ""

can derive "a" "a, a" "a, a, a" etc.

Question

Show the derivation of "a, a" from this grammar (A is start symbol)

```
A ::= a B
B ::= , a B | ""
```

Answer

Show the derivation of "a, a" from this grammar (A is start symbol)

A ::= a B
B ::= , a B

A → a B → a , a B → a, a

Question

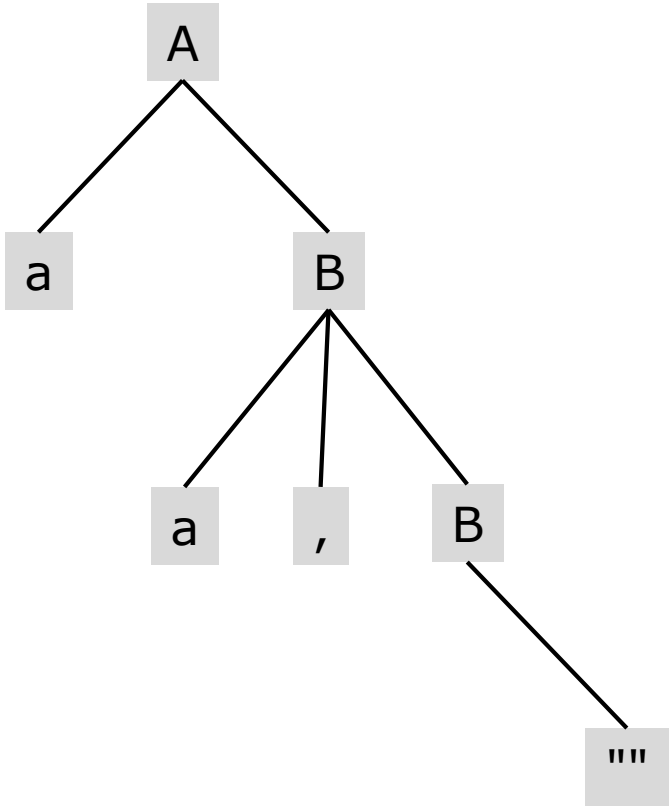
Draw a parse tree to show that "a,a" is in the language defined by this grammar
(A is start symbol)

```
A ::= a B
B ::= , a B | ""
```

Answer

Draw a parse tree to show that "a,a" is in the language defined by this grammar
(A is start symbol)

A ::= a B
B ::= , a B | ""



Question

Can you derive "(()())" from the following grammar?

```
S ::= S ( S ) S  |  ""
```

Answer

Can you derive "(()())" from the following grammar?

```
S ::= S ( S ) S  |  ""
```

Yes.

```
S → … → ( S )
  → ( S ( S ) S )
  → … → ( () S )
  → ( () S ( S ) S )
  → … → ( () () )
```

Question

Describe the language that is defined by the grammar.

```
S ::= S ( S ) S  |  ""
```

Answer

```
S ::= S ( S ) S  |  ""
```

This language contains all strings of matching parentheses (including the empty string).

Question

Rewrite this grammar to make it suitable for predictive parsing.

```
S ::= S0 | S1
S1 ::= 1 S0 | ""
S0 ::= 0 S1 | ""
```

Answer

Rewrite this grammar to make it suitable for predictive parsing.

```
S ::= S0 | S1
S1 ::= 1 S0 | ""
S0 ::= 0 S1 | ""
```

A solution:

```
S ::= 0 S1 | 1 S0 | ""
S1 ::= 1 S0 | ""
S0 ::= 0 S1 | ""
```

Question

Rewrite this grammar to eliminate left recursion:

```
S ::= S "," a
    |   a
```

Answer

Rewrite this grammar to eliminate left recursion:

```
S ::= S "," a
    |   a
```

A solution:

```
S ::= a R
R ::= "," a R | ""
```

```
A ::= A α | β
```

derives the same strings as:

```
A ::= β R
R ::= α R | ""
```

File Management Review

File Management

- Main goal: virtualize persistent storage
- Storage is virtualized as files and directories
- Main technical problems:
  - how to avoid having OS know low-level details of each specific I/O device?
  - how to store all file and directory information as an on-disk data structure?
  - how to support very large files but handle the common case of small files efficiently?

Question

What CPU instructions are used in the x86 architecture to read or write the registers in a device interface?

a) special 'in' and 'out' instructions

b) normal instructions used for reading and writing memory

Answer

a) special 'in' and 'out' instructions

Question

A hard drive has a transfer rate of 120 MB/sec and access time of 10 ms.

How long will it take to perform a single, sequential read of 60 MB?

Answer

Once the gas starts flowing, it will take 0.5 sec to transfer the data. So the answer is

10 ms + 500 ms = 510 ms

In this case the access time overhead is small.

Question

(T/F) In Linux, disk requests are serviced in a first-come, first-served manner.

Answer

F

This would be inefficient.

We discussed disk scheduling policies like:

- shortest seek time first
- elevator scheduling

Question

You executed the following commands:

```
$ echo blecch > foo
$ ln foo bar
$ rm foo
```

What happens when you execute this command?

```
$ cat bar
```

a) error

b) nothing is output

c) 'blecch' is output

Answer

'blecch' is output

bar is a hard link to foo

bar and foo are two names for the same file

Bonus: hard homework/exam problems

Exam 1, problem 15

Three jobs arrive to be processed at almost exactly the same time.

The first job to arrive takes 40 seconds to run, the second job takes 10 seconds to run, and the last job takes 70 seconds to run.

Suppose each job is run until it is finished.

What is the average turnaround time under round-robin scheduling? (assume very small time slice)

sum of turnaround times is 1*70 + 3*40 + 5*10 = 240
answer: 240/3 = 80

For processes A1, …, An (with A1 >= A2 >= … >= An),

the sum of the turnaround times is:

1*A1 + 3*A2 + 5*A3 + …

Homework 3, problem 1c

What does the ‘wait’ function return when it succeeds?

a) 0,

b) process ID of the terminated child, or

c) process ID of the parent.

answer: b (see man page)

Homework 3, problem 1d

What happens if function wait is called by a process with no children?

a) the wait call returns immediately,

b) the wait call never returns,

c) the behavior of wait in this case is not defined.

answer: a (see man page)

Exam 1, problem 17

Calculate average turnaround time for two jobs running under "shortest time to completion first".

Job A arrives at time 0, and takes 30 seconds to run.

Job B arrives at time 10, and takes 10 seconds to run.

What is the average turnaround time?

Answer: 25

Exam 1, EC problem 2

What does execvp() return? _____

Answer: nothing if successful; else an error code

Exam 1, EC problem 2

Suppose we're using a multi-level paging scheme. The virtual address space is 1 KB. The page size is 32 bytes, so there are 32 pages in the virtual address space. Also, a page table entry requires 4 bytes of space, so 8 page table entries fit into 1 page. How many entries in the page directory? _____

Answer: 4

Extra bonus content: the stack

You know that the heap is used for dynamically-allocated storage. What is the stack used for?

It used to support the use of functions:

- function arguments are pushed on the stack (sometimes)
- the return address is pushed on the stack
- local variables of a function are stored on the stack
- return values are stored on the stack (sometimes)

A stack is needed because functions can be called recursively.