CS 350 L#16 Tuesday, March 9, 2009

Bitwise ops (cont'd) -

Multiplication

Example 10×11

$= (8+2) * (8+2+1)$

$= (2^3+2^1) + (2^3+2^1+2^0)$

$= 2^3 \cdot 2^3 + 2^3 \cdot 2 \cdot 1 + 2^3 2^0 + 2^1 2^3 + 2^1 2^1 + 2^1 + 2^0$

$= (8+2^3 + 8 \times 2^1 + 8 \times 2^0) + (2 * 2^3 + 2 * 2^1 + 2 + 2^0)$

$= (8 << 3) + (8 << 1) + (8 << 0) + (2 << 3) + (2 << 1) (2 << 0)$

$= (88) \quad + (22) = 110$

Hexadecimal + octal constant

0x 177 → hex | 0001 | 0111 | 0111 | $1 \times 16^2 + 7 \times 16^1 + 7 \times 16^0 = 375$

= 375

0177 → | 001 | 111 | 111 | $= 1 \times 8^2 + 7 \times 8^1 + 7 \times 8^0 = 127$

Operations on bits

setting a bit

$i = 0x0000;$

$i \mid = 0x0001 \quad =$ | 0000 | 0000 | 0001 | 0000 |

or operator

clearing a bit | 0 | 0 | 0001 | 0 | 0x0010

i = 0x00ff; | 0 | 0 | 1111 | 1111 |

i &= 0x0100 | 0000 | 0000 | 0001 | 0000 |

$i = 0x0ff$ | 0 | 0 | 1111 | 1111 |

$i \mathrel{\&}= \sim(1 << j)$ | 0 | 0 | 0 | 1000 |

$j = 3$

shift 1 = 3 time | 1111 | 1111 | 1111 | 0111 |

Simple Encryption using XOR

- encryption stage - process message with OK
- decryption stage

Encrytion : XOR(Message, Key) → encrypted Message

Decryption: XOR(encrypted Message, Key) → original message

Key = "&" = |0|0|1|0|0|1|1|0|

Use Key '&'

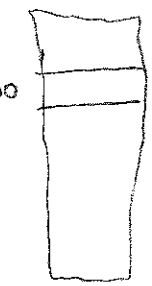
Message - T R U S T N O T H I M
- T S U R H I R N O K

Th
F 12
S 13
S 14
M 15
T 16

Missed Thursday, March 11, 2010

Tuesday, March 16, 2010 Lecture #18

Topics - Random Access Files
Command Line
Operating Systems
→ introduction
→ history



1000

C Programming

Random Access File

Data unformated stored as binary

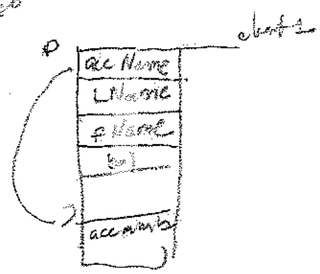
points 30

Creating
f write
fread
fwrite (& numbe, size of (int), 1, myPtr);

File to transfer to or from

fread - reads a specific number of bytes
fread (& client, size of (struct clientData), 1, myPrinter)

fseek writing randomly to a Random Access
fseek (my Ptr, offsets, symbolic_ constant)
myPtr → pointer to File
offset → file position pointer (0 is th first location)
symbolic - constant
seek cut - start at begin
seek_cur start at loc



struct clientdag *p
P = & client 1
P = P + 1

Command Line

```
int main()
{
    .
    .
    .
  return 0
}
```

Pass argument

$cp f1.text. f2 text

$cp -ip ~~f1text~~ f2~~text~~ Look at 14.1 →

int main (int argc, char *argv[])

Struct passed as values
Array passed as variable

k=20

8	16	4	2	1
1		1	0	1

10100

k<<2 shift value of k

Operating Systems

- history
- processes/threads
- Interprocess communication
- Deadlock
- Files
- Memory

Moore's Law entry in Wikipedia

10^{+10}

In vacuum

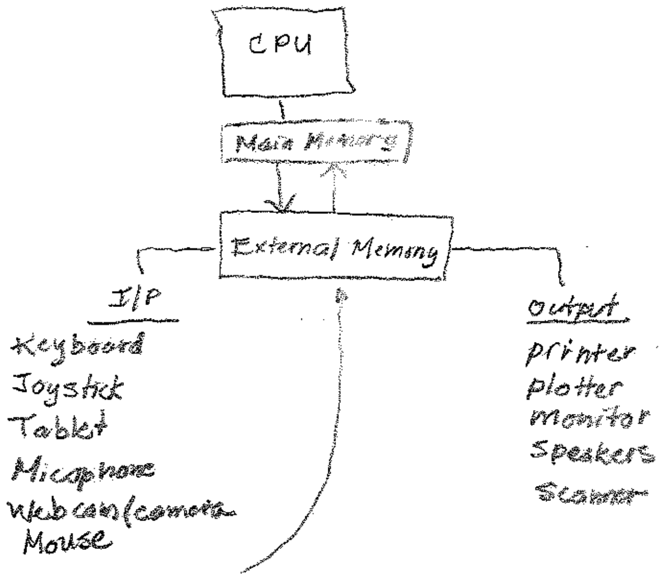
5/1979 → .1 GB
5/2007 → 105 GB

Parkinson's Law - The amount of money causes our problems to grow.

Source Lines of Code

1993	Windows NT 3.1	4-5 Million
1994	Windows NT 3.5	7-8
1996	Windows NT 4.0	11-12
2000	Windows 2000	more than 29
2001	Windows XP	40
2003	Windows Server 2003	50 million

Typical Computer Hardware

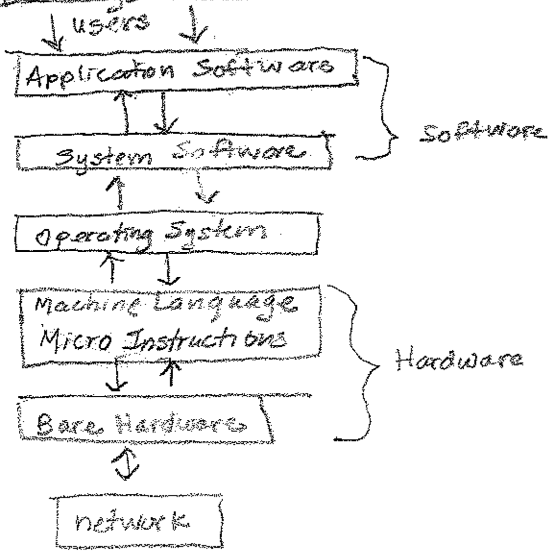


Input-Output Devices
network adaptor
touch screen
modem
haptic devices

The OS is taken to be a manager of computer resources so what do we expect from a "good" manager?

- Organization of resource
- scheduling
- fair allocation
- Priorities
- response/action
- conflict/resolution
- efficiency
- protection + security
- Utilization

Computer System Hierarchy



Now is the software engineering

Operating system services

System side
resource allocation
accounting, protection

User side
program execution
file manipulation
I/O operations
communications
error detection

Operating System boundaries

The operating system is taken to be all software that

Os programs run in:
Kernel Mode
→ protected by hardware

Other programs run in user mode

History

Babbage's analytical engine – no OS
(1938)
1st generation (1945-1955) Vacuum tube, Plugboards
later punch cards No Os

2nd generation (1955-1965) Transitor-based
Fortran management System & IBSYS
only one program in memory

1950's batch processing

60's multiprogramming, spooling, PDP-8

70's Time sharing, real time sys, unix

80's Parallel processor, PC's

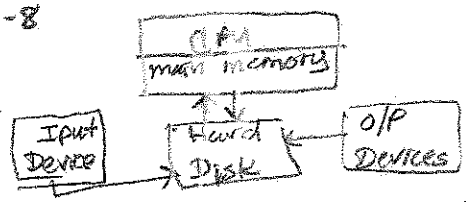
1990's Networks, Internet Linux

2000's Security and Fault tolerance
Wearable computers

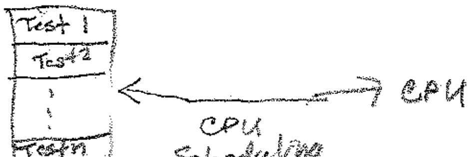
2010's ?

Batch processing – speed disparity

Spooling
Put input into hard disk



Speed disparity b/w cpu + I/O Devices
Multiprogramming – divide memory into separate units



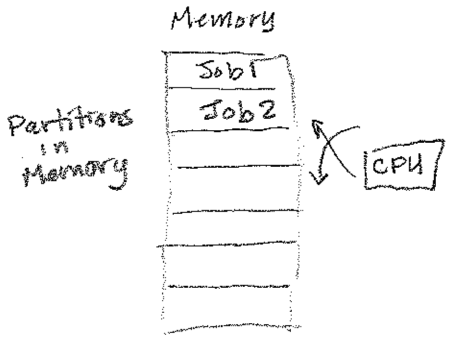
CPU Scheduling → main memory to CPU

Job Scheduling
Controls which jobs access the main memory

Time sharing
This involves users
Time slice –

Tuesday, March 23, 2010

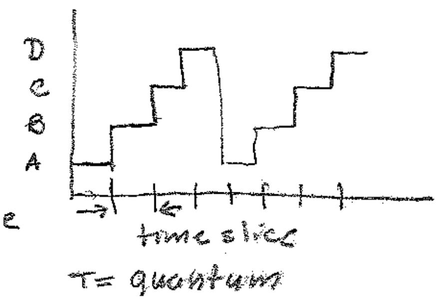
Topics -
- Time Sharing
- Processes
  - Intro
  - Creation
  - Process Trees



Multiprogramming -
It was ~~taking~~ too long to upload from the hard disk to main memory.

Time Sharing
Divide the CPU into time slice and rotate jobs thru until a job completes
Needed to accommodate both interactive jobs and CPU-intensive jobs

use round-robin to distribute CPU time



must be careful about the length of T - too long and interactive programs will hang too long

Let T = time slice in units of time
S = start-up time to load into CPU
P = actual time for processing or executing program

Utilization, $U = \frac{P}{S+P}$

if P is small $\frac{1}{\left(\frac{S}{P}\right)+1}$

if $(S+P) < T \Rightarrow$ then complete in one time slice

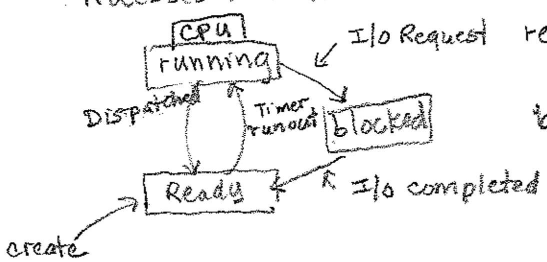
4th generation (1980-1990)
multiple processors

Processes
myProgram.c $\rightarrow$ myprogram.ex $\xrightarrow{\text{load}}$ myProgram (In memory) $\xrightarrow{\text{time slice}}$ $\boxed{\text{CPU running}}$

when program is in CPU running $\Rightarrow$ processes

unix $\rightarrow$ \$ myProgram.ex

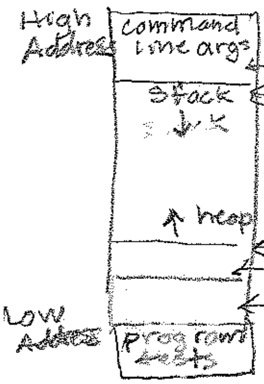
Processes have multiple states



ready - temporarily suspended by CPU to let others run
blocked (sleeping) - unable to run - waiting for external eg I/O

MRB 243
6:30-8:30

Image of a Process (in memory



argc, argv, enenum
← enumerated variable
← activation needed for function calls (return addresses, parameters saved registers
← allocation from malloc family
← unitialized static data
← initilized data

malloc Family -

Mu

P= pick up memory
malloc (n); n → in bytes

P= calloc (n) P - Pointer

P= realloc - (P, m
↑ size

Two Programs

A

Takes space immediately
↓

```
int myArray[50,000] = {1,2,3}
int main ()
  {
  myArray[0] = 3;
  return 0;
  }
```

↑ must be in memory

This only uses the space when we need it

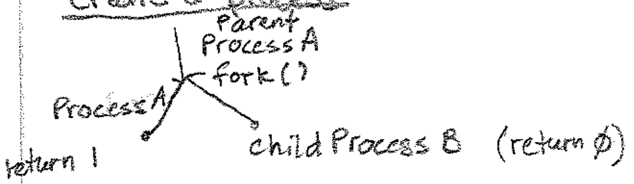
B

```
int myArrary[50,000];
int main( )
{
 myArrary [0] = 3
 return 0;
```

Go back to system
$ LS-L /proc find a lot of information

Create a process

$ myProgram ↵



Three important issues
asynchrony - no control over time issue
communication
consistency

CS350

Tuesday, April 6, 2009 Lecture #22

The shell
OS System
IPC
Signals

Multiprogramming → CPU + I/O
Time sharing → CPU + human IO
Spooling → CPU + Secondary stage
Batch processing → Interactive jobs & computational - intens

Exec Family

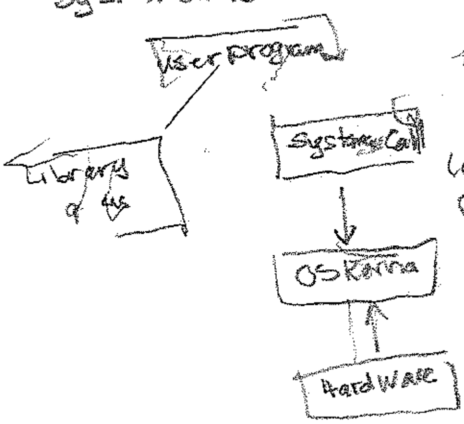
Ishell

$> list-1.d

Running in background
$> my KnightsTour > old File . tex & ;
output reduction
$>
$>
$cat < my File.txt
$cat < my File, tex > out put. txt ← I/O redir

p.60
Process control
Device Mana
Information

System calls



(open)
close
Read()
Write

Return -1 for error

CTRD
if (fptr

Book

Interpr

Interprocess Communications p.24
Don't allow two resources to use the
Sync

Communications process.

MRB 243

Signals
One meta
Hardware - divide by 10
OperatingSA
Hardware Drive can not tolerate division by zero.

I kept falling asleep -
Notes incomplete
for 4/6/10

Thursday, April 8, 2010

Signals
Race Conditions
Semaphore

Signals
Interprocess communications (IPC)
need a way to ensure data signals

Other signals: hardware - divide by 0
operating system - file size exceeds
child → parent

Actions default - usually cause the process to terminate
Ignore - SIG STOP (stop process from executing) SIG KILL
Catch the signal - executes

$ > kill 2785 running too long -
> kill -23 2789 stop
kill - 25 2789 continue

list by using > kill -l

SIGINT - Interrupt
SIG not in correct place in memory

RAISE send signal to yourself

struct sigaction act; ← include with all programs

$ > ps
sh 2789

stdin → 0
stdout → 1
stderr → 2
infile 3
outfile - 4

struct point
{
int x;
int y;

act sa_handler =
signal processor = catch_ctrl c_
sig emptyset
act: sa. flag_ flags = 0

basic communication techniques
most signals call for terminate

signals can not carry data.

Shell ⇒ 2789 must know

multiprocess ?

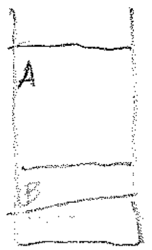
Semaphores + shared Memory

shared int account
{
/* deposit with race condition */
void deposit(int money)
int balance = account;
account = balance + money

Account = 5
Process A: (-1)
Process B: (+4)

Final Result
Process A: +4
Process B +6
Final +6

Time	Process A	Process B
T_0	int balance = account	
T_1		int balance = account
T_2	account = balance + money	
T_3		account = balance + money
T_4	} end of program	
T_5		



More race conditions shared int lock

Add lock to prevent shared services.
void deposit(int money)
while (lock == 1); /* busy wait */
lock = 1
balance = account;
account = balance + money
lock = 0;

locks may not work for multiple users

~~why~~ bad because
because of time slice problems
stuck in while loop -- can't use
efficiency of memory computer
timeslice - will keep using ~~efficien~~ time

Critical Section problem
mutual exclusion - only one process
progress - the second
Bounded waiting - how long

Tuesday April 12, 2010

Topics . Semaphores
Message Passing
Pipes

Mutual Exclusion -

Progress - Holding Time w/o

Bounding Waiting - No process is postponed immediately

" One pigeon in one hole "

Disabling interrupt signal → go back to time slice

Lock variable -

Strict alteration

high priority process is waiting for low priority programs until it

TSL Intruction hardware - can't do anything else

h

Semaphores

Introduce by

A flag used to raise best

→ semaphore

These process is atomic - it cannot be interrupted

two bits - yes or no

Down → puts process to sleep if semaphore is zero

up →

atomic

Semaphores

```
Down (s)
{
if (s ≤ 0) sleep
   s = s-1
}
```

```
up (s)
{
s = s+1;
wake up sleeping process
}
```

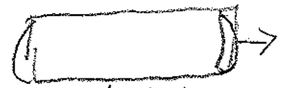
semaphore example pg37 of handouts

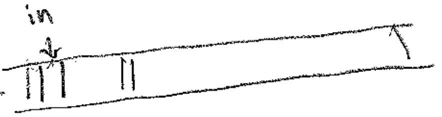
```
share binary semaphore mutex = 1;
void deposit (int money)
{
int balance;
down (& mutex);
int balance = account;
account = balance + money;
up (& mutex);
}
```

sem - wait ()
sem - post ()

which process to wake up? high priority first
OS keeps a queue w/ time stamp
OS - lottery

Semaphore synchonization
Handout p.38
called producer-consumer
producer doesnt over load consumer   buffer over runs
consumer - realize must use what producer
producer can't provide enough   starvation

speaker 
want it to have streat line
do not over load
do not under load -



always put down (empty)
before the ~~mutex~~ down (mutex

april 15, 2010
Topics: Message Passings
Pipes
Threads (intro)
Review for Test 2
Note on Assign 3

A ⟶ B   B not available

Message Passage
two primitives: send (write)
receive (read)

Buffered - store
unbuffered - the message will be lost
Blocking - A cannot not do any thing until the process notifies the message was not ~~sent~~

Message Boundery Preserving

A [3] [2] [1] ⟶ [3] [2] [1] B_1 preserved - Packets Preserved
[3|2|1] B_2 non-preserved

p.42 HALF Duplex - Communication one-way only
Full Duplex - Two-way

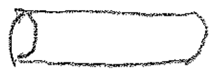
Reliable - The order of the messages sent is received in the same order
Unreliable - Order is not perserved
Internet

Networking between two processors in seperate location process synchr

A cknowledgement / re-transmissions
Naming / addressing
shell.csee.wvu.edu → IP address
Authetincation
Performance - overhead - the above takes more time

Pipes
From one process to another process.

Named files	Unnamed Files	Both
File Named	general file description	Message boundaries not preserved
Outside process can see	int fdes [2]	BUFFERED
Persist after processes have ended	pipe (fdes)	FIFO
	fdes[0] } each end / fdes[1] } of pipe	Blocking or non-Blocking
	Pipe disappears after processes finish	

fdes[0]  → fdes[1]

```
int main( )
  {  int msg; res;
  int fdes [2];
   = Pipe (fdes);
pid= fork ( );
if (pid = = 0)          msg = getpid();
  write (fdes [0], msg, sizeof (msg));   ← child writing
else
  read (fdes [1], & res, size.of (msg));
```

parent PID → (fork)
Child → (if)
parent waiting → (read)

Child process aquires all the files

(parent)		child
stdin	0	"
std out	1	"
stderr	2	"
fdes [0]	3	"
fdes [1]	4	"

write b/4 read

#49 default:
parent → read fdes [0]

receive / send structure — for assignment

The example is a deadlock → program hangs out
No easy way to determine
can't communicate between processes on different machines.

Read up on threads

Exam Review on Tues April 20, 2010

C part
- Strings
- Structures
- Files
- Bitwise operators

OS
- intro + history → concepts in Os
- shell
- process
- system calls
- signals — to pg 49
- race conditions
- semaphores
- pipes + message passings
- IPC(?)

Content:
- Concepts
- T/F
- Errors in programs
- Results
- Execution sequence

Assignment 2

Notes on Assignment 3

Need Assignment 2 to work for image

Divide image into 4 parts → send to different processors

1. image processing
2. How to distribute the job to multiple processors or threads
3. How each processor will perform its analysis on its own part of the image (Assignment 2)
4. How the different processors will send back their results
5. How the final result is organized using the results sent by individual processors.

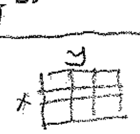
create the

for (p=0, p<5
fork()
if (PID == 0)
break
}

counter = 0
for (i =
for (j
for (x
for (y
{
small Block [ ]
}
counter ++

block label = counter % P
if (process Label == block Label)
;

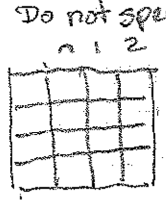
children



nxn = 3x3

analyze small Block
return [m, end
send back results to parent

Do not specify # of processor



assign a block
process Label 0, 1, 2
pixel ID % #p