

CSE-410-002 Final Exam Review Sheet

Scott Warren

December 9, 2007

1 Chapter 5 - Concurrency: Mutual Exclusion and Synchronization

Semaphores w/Monitors: wait(x){
x.count-;
if(x.count < 0)
cwait(c);
}
signal(x) {
x.count++;
if(x.count ≤ 0)
csignal(c);
}

Requirements for a soln to CS: Mutual Exclusion:

Progress: If no process is executing its CS while some process wishes to enter, any process that requires entry should be granted such w/o delay

Time: Process remains in CS for finite time

Delay: A waiting process cannot be delayed indefinitely

Semaphore: wait(s){ s.count-; if(S.count < 0){
block process
place process in queue } }
signal(s){ s.count++; if(s.count ≤ 0){ remove a
process from queue
place process on ready list } }

: When S.count < 0 the number of processes waiting is -S.count-

Deadlock: A situation in which two or more process are unable to proceed because each is waiting for one of the others to do something

Livelock: A situation in which two or more processes continuously change their state in response to changes in the other process(es) w/o doing any useful work

Race Condition: A situation where multiple processes read and write shared data items and the final results depends on the relative timing of the execution

Starvation: A situation in which a runnable process is overlooked infinitely often by the scheduler although it is able to process it is never chosen

xchg for ME:

k=1;
repeat xchg(x,b) until k=0
CS
b=0
RS

Peterson's Algorithm:

Process P_i
flag[i]=true // tell the other process that it wants CS
turn=j // let the other other process go first
while(flag[j] & turn=j) // wait until the other process is done
CS
flag[i]=false // done with CS
RS

Bakery Algorithm: Process P_i

repeat
choosing[i] = true; // this process is choosing a

```

number
number[i] = max(number[0]...number[n-1])+1 // get
the largest number
choosing[i] = false; // the process has the highest
current number
for j = 0 to n-1 {
while(choosing[j]){ // make sure other process isn't
choosing
while(number[j]!=0 and (number[j],j);(number[i],i)
{} // make sure the other process doesn't have
priority
}
CS
number[i] = 0; // This process is done
RS
forever

(a,b) < (c,d) - > if a<c or if a =c and b < d

Reader/writer: Readers have priority: x=1
usem=1
readcount=0
void reader(){
while(true) {
semWait(x)
readcount++
if(readcount == 1)
wait(wsem)
signal(x) ReadUnit() wait(x) readcount-
if(readcount==0) signal(wsem) signal(x) }
}
void writer() {
while(true){
wait(wsem)
writeunit()
signal(wsem)
}
}

Finite Circular Buffer P/C: s.count =1
n.count = 0
e.count = k
Producer
repeat
produce v
wait(E)
wait(S)

append v
signal(S)
signal(N)
forever

Consumer
repeat
wait(N)
wait(S)
w=take()
signal(S)
signal(E)
consume(w)
forever

reader/writer:writers have priority: void
reader() {
wait(z)
wait(rsem)
wait(x)
readcount++
if(readcount==1)
wait(wsem)
signal(x)
signal(rsem)
signal(z)
read()
wait(x)
readcount-
if(readcount == 0)
signal(wsem)
signal(x)
}

void writer(){
wait(y)
writecount++
if(writecount==1)
wait(rsem)
signal(y)
wait(wsem)
WRITE()
signal(wsem)
wait(y)
writecount-
if(writecount == 0)

```

```

signal(rsem)
signal(y)
}

```

2 Chapter 6 - Concurrency: Deadlock and Starvation

Deadlock: Perminate clocking of a set of processes that either compete for system resource or communicate with each other.

Reusable Resource: Used by one process at a time and not depleted by that use i.e. processors, I/O channels, main/secondary memory, files, databases, semaphores

Consumable Resources: Created (produced) and destroyed (consumed) by a process e.g. Interrupts, signals, messages, and info in I/o Buffers

Three Approchs to Deadlocks: Prevention: Adopting a policy that eliminates one of the conditions

Avoidance: Making appropriate dynamic choices based on current state of resource allocation

Detection: Detect deadlock and take action to recover

wait-die: If T2 is older, it is blocked until T1 releases R. If T2 is younger than T1 it is restarted with old time stamp

wound-wait: Immediately grants request of older process by killing younger process. Older process never has to wait for younger process

Solving Dining Philosophers Problem: Prevention - Ensure no circular wait
avoidance - ensure resource allocation does not lead to unsafe state.

Four Conditions for Deadlock: Mutual Exclusion:

Hold and Wait: a process may hold allocated resources while awaiting assinment of other resources

No Preemption: No resource can b e forcibly removed from a process holding it

Circular Wait: A closed chain of processes exist such that each process holds al least one resource needed by the next process in the chain.

Deadlock Prevention: No ME: Cannot be disallowed

Hold and Wait: can be prevented by:

A process requests all resources at a time

Blocking process until all request can be granted
Inefficient

No Preemption: If a process hold certain resources is denied for a futher request, that process must release it's original resource

Circular Wait: Define a linear ordering of resource types

may be inefficient slowing down processes and denyin resources access unnecessarily

Stratagies afer Deadlock Detected: Abort all deadlocked processes

Backup dadlocked processes to some previously defined checkpoint and restart all processes - orignal deadlock may occur

Successively abort deadlocked processes until no longer exists

successively preempt resources until deadlock no longer exists

Selection Process for deadlocked processes:

Least amount of processor time used so far

least number of lines of output

most estimated time remaining

least total resources allocated so far

lowest priority

Baker's Alogirthm: State: currently allocation of resources to process

Safe State: a least one sequence that doesn not result in deadlock

Unsafe State: a state that is not safe

Resource: vector reresenting the total number of each type of resource in system

Available: vector representing maximum number of each type of resource no allocated

Claim: matrix w/ maximum number of resource a process can request

Allocation: matrix w/ current allocation

Need: Matrix w/ difference between Claim and

Allocated matrices

Request: vector representing a request for additional resources for a process

3 Chapter 7 - Memory Management

Need for Memory Management: OS and hardware need to accommodate multiple processes in main memory

If only a few processes can be kept in main memory, then much of the time all processes will be waiting for I/O and the CPU will be idle

Hence, memory needs to be allocated efficiently in order to pack as many processes into memory as possible

In many systems, the kernel occupies some fixed portion of main memory and the rest is shared by multiple processes

Memory Management Requirements: **Relocation:** Programmer does not know where the program will be placed in memory when it is executed

Protection: Processes should not be able to reference memory locations in another process without permission

Sharing: Allow several processes to access the same portion of memory

Logical Organization: Programs are written in modules

Physical Organization: Secondary memory is the long term store for programs and data, while main memory holds programs and data currently in use

Fixed Partitioning: Partition main memory into a set of non overlapping regions called partitions

Dynamic Partitioning: Partitions are of variable length and number

Dynamic Partitioning Placement Algorithm:

Best-fit algorithm: Chooses the block that is closest in size to the request - Worst performer overall

First-fit algorithm: Scans memory from the beginning and chooses the first available block that is large enough - Fastest

Next-fit: Scans memory from the location of the last placement

Buddy System: If a request of size s such that $2^{U-1} \leq s < 2^U$, entire block is allocated

Relocation: A process may occupy different partitions which means different absolute memory locations during execution (from swapping)

Logical Address: Reference to a memory location independent of the current assignment of data to memory

Relative Address: Address expressed as a location relative to some known point

Physical Address: The absolute address or actual location in main memory

Base register: Starting address for the process

Bounds register: Ending location of the process

4 Chapter 8 - Virtual Memory

: Memory references are dynamically translated into physical addresses at run time

Advantages of Partial Loading: More processes may be maintained in main memory

A process may be larger than all of main memory

Principle of Locality: The key to the success of this approach to memory management is that instruction and data references in a program sequence tend to cluster

Virtual Memory: For better performance, the file system is often bypassed and virtual memory is stored in a special area of the disk called the swap space

Possibility of Thrashing: To accommodate as many processes as possible, only a few pieces of each process is maintained in main memory

But main memory may be full: when the OS brings one piece in, it must swap one piece out

The OS must not swap out a piece of a process just before that piece is needed

Paging: Each process has its own page table and

a page table contains the frame number of the corresponding page in main memory

Inverted Page Table: Page number portion of a virtual address is mapped into a hash value

Page Size: Smaller page size, less amount of internal fragmentation but Smaller page size, more pages required per process

Different Memory Management Policies:

Fetch Policy: Determines when a page should be brought into memory

Placement Policy: Determines where in real memory a process piece is to reside

Replacement Policy: Page removed should be the page least likely to be referenced in the near future

Optimal policy: Selects for replacement that page for which the time to the next reference is the longest

Least Recently Used (LRU): Replaces the page that has not been referenced for the longest time

First-in, first-out (FIFO): Treats page frames allocated to a process as a circular buffer

Clock Policy: Additional bit called a use bit
When a page is first loaded in memory, the use bit is set to 1

When the page is referenced, the use bit is set to 1

read or written passes through the processor

Direct Memory Access: Processor delegates I/O operation to the DMA module

DMA module transfers a block of data directly to or from memory

Processor interrupted only after entire block has been transferred

The processor is only involved at the beginning and end of the transfer

I/O Buffering: Stores information in a buffer - only one process can be reading/writing to buffers

Double Buffering: Two buffers are used so one can be read while the other is writing

Circular Buffering: More than two buffers are used

Disk Performance Parameters: **Seek time:** time it takes to position the head at the desired track

Rotational delay: time it takes for the beginning of the sector to reach the head

5 Chapter 11 - I/O Management and Disk Scheduling

Programmed I/O: I/O module performs the action, not the processor

Sets appropriate bits in the I/O status register

No interrupts occur

Processor checks status until operation is complete

Process is busy-waiting for the operation to complete

Interrupt-Driven I/O: I/O command is issued

Processor continues executing other instructions

Processor is interrupted when I/O module ready to exchange data

No needless waiting

Consumes a lot of processor time because every word

Disk Scheduling Policies: **First-in, first-out (FIFO):** Fair to all process but appears random if there are many processes

Priority: Don't optimize disk access but meet other goals

Last-in, First-out: Device is given to most recent user so there should be little arm movement

Shortest Service Time First: Select I/O request that requires least arm movement

SCAN: Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction

C-SCAN: When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again

:

:

6 Chapter 12 - File Management

Preallocation: Need the maximum size for the file at the time of creation

Dynamic Allocation: Allocate space to a file in portions as needed

Types of files: Ordinary

Directory

Special

Named

Links

Symbolic links

Inodes: A control structure that contains the key information needed by the OS for a particular file