1 Problem I

PART A

struct {
    int count;
    Queue q;
    int lock;
} SemT;

void init(SemT *s, init val){
    s->count = val;
    s->lock=0;
}

p(SemT *s){
    while(TestAndSet(&(s->lock))!=0);
    while(count==0){
        put myTCB on s->q;
        s->lock=0;
        RunNextThread;
        while(TestAndSet(&(s->lock))!=0);
        s->count--;
        s->lock=0;
    }
}

v(SemT *s){
    while(TestAndSet(&(s->lock))!=0);
    s->count++;
if(s->q not empty){
    MakeRunnable(dequeue TCB);
}
}

PART B

typedef struct{
    SemT RWait;
    SemT WWait;
    int Rcount=0;
    int Wcount=0;
    int RWaitCount=0;
    int WWaitCount=0;
    SemT lock;
} RWLock;

void init(RWLock *l){
    init(&l->RWait,0);
    init(&l->WWait,0);
    init(&l->lock,1);
}

void LockRead(RWLock *l){
    p(&l->lock);
    while(1){
        if(l->Rcount==0){
            Rcount++;
            v(&l->lock);
            return;
        }
        l->RWaitCount++;
        v(&l->lock);
        p(&l->RWait);
        p(&l->lock);
        RWaitCount--;
    }
}

void LockWrite(RWLock *l){
p(&l->lock);
while(1){
    if(l->Rcount==0 & & l->Wcount==0){
        wcount++;
        v(l->lock);
        return;
    }
    l->WWaitCount++;
    v(&l->lock);
    p(&l->WWait);
    l->WWaitCount--;  
    return;
}
  }

void unlockRead(RWLock *l){
p(&l->lock);
l->Rcount--;  
if(Rcount==0 & &WWait>0){
    v(&l->WWait);
}  
else{
    v(&l->lock);
    return;
}
  }

void unlockWrite(RWLock *l){
p(&l->lock);
l->Wcount--;  
int i=l->WWaitCount;
while(i-->0){
    v(&l->WWait);
}
    i = RWaitCount;
while(i-->)0{
    v(&l->RWait);
}
    v(&l->lock);
2 Problem II

In C there is no memory protection within process, so pointer can go anywhere inside the process address space. Consequently private stack for thread is not enforced.

In Java, however, one can get only safe reference, so language checks the safety.

3 Problem III

The Linux scheduler is just a variant of multilevel feedback queue with two queues: one for interactive and one for batch jobs. So basically in Linux there are only two kinds of jobs: interactive and batch with the characteristics that batch jobs are CPU-bound and interactive jobs are IO-bound. The scheduling decision is made on the value of goodness which basically based on the counter value associate with each process. These counter value is decremented after each tick the process is running on the processor, so the batch jobs will consume the ticks more quickly then the batch jobs, giving the interactive jobs more tickets meaning more goodness value so higher priority at the scheduling decision time. To prevent the starvation of batch jobs, the counter value is reset after every process in run queue has the counter value of 0.

[[Thu: The Linux scheduler doesn’t necessary have just two queues. In general, not just for Linux, for general-purpose processing systems, historically, there has been only two types of applications, interactive and batch.]]

4 Problem IV

Both BTV and Lottery scheduling address the problem of Linux scheduler that it distinguishes between only two kinds of jobs: interactive and batch. However, with the emergence of multimedia this model of only two jobs breaks, and the Linux scheduler performs bad in presence of this type of application. So BTV and Lottery scheduling address this problem by avoiding the static assumption of having only two kinds of jobs, but instead making the scheduling decision by watching their CPU usage. Both methods are complementary, meaning you can implement one using another. The
difference between two approach is that BTV is deterministic and Lottery scheduling is probabilistic. The notion of lottery can apply to a lot of areas in resource management, however the BTV can apply only to those area where the notion of time is present.

[[Thu: I wouldn't claim that the two methods can be used to implement each other precisely. This is because one is completely deterministic and one is probabilistic. What I think is that the two seem equivalent in power, meaning that they can be used to implement pretty much the same class of scheduling policies, although the details of the implementation would different somewhat. Also, it's not quite that the model of “two types” of jobs breaks; it's that statistically, it is now difficult to judge whether an application is interactive or batch based on its pattern of CPU usage. A multi-media application is interactive in that some user is using the sequence of outputs as they are generated (e.g., watching the frames of a movie) but it is CPU intensive and so may look just like a batch application.]]

5 Problem V

Virtual memory supports private address space by the fact that process generates only virtual addresses which have to go through a translation process that is beyond the control of process, so it is impossible to generate an address that maps to different process physical memory space unless it is explicitly asked for. The U-net takes the advantage of setting up the endpoints by kernel in the virtual address space, and the translation of address happens at the card with aid of kernel. So U-net is as safe as the virtual memory system.

6 Problem VI

FFS assumption is greater bandwidth but not seek time and rotational latency. Must optimize both read and write traffic. It is achieve by having bigger blocks, and better organization like placement of inodes and blocks.

LFS assumption is faster disk in means of bandwidth, rotational latency and seek time but just fit in to improvement of technology trend. The main assumption is that memory is much bigger, so it can satisfy most of read traffic. It also makes assumption about the temporal locality of reads. So improvement in file system can be achieve by optimizing the write traffic.
7 Transaction

Atomicity: Either all of the operations in a transaction are performed or none of them are, in spite of failures. Achieved using two-phase commit.
Consistency: The execution of interleaved transactions is equivalent to a serial execution of the transactions in some order.
Isolation: Partial results of an incomplete transaction are not visible to others before the transaction is successfully committed. Optimistic transaction allows intermediate results to show, but when abort every who sees the result have to abort also. The property of consistency and isolation are achieved by using two-phase locking.
Durability: The system guarantees that the results of a committed transaction will be made permanent even if a failure occurs after the commitment.

2-phase locking: has to phases: acquire and release. During the acquire phase if once release a lock can never acquire another lock. In strict two-phase lock once if release a lock every lock has to be released. During release phase all the locks have to be released at the end of transaction, either abort or commit.
This protocol does not prevent from deadlock, ordering of locks prevents deadlock.

8 Birman Paper

• process group, virtual synchrony

• motivations for group:
  – replication for reliability
  – multiple different things together. Natural flow and dependency of different processes.

• Virtual synchrony:
  – View events in the same order. It is natural for set of related things, this gets ordered to the casualty, and all members agree on the order of one-to-many messages. This one-to-many messages cannot interleaved, thus producing a casual ordering between them.
  – Multicast <---> Total ordering The application decides what are related. So transaction must build on top of this. The underneath layer can provide th concurrency based on that.
- Exception model is not clear
- Interaction of group membership and the delivering of messages: either deliver to everyone or none. If one node fails, the message has to be reset. Messages carries something that tells about the ordering eg. vector of version of virtual clock.