In-Class Problems: forkprint

Here is a threaded program:

```
main() {
   helper(NULL,0);
}

void
helper(void *p, unsigned long i) {  /* parameter p is not used */
  if (i < 3) {
     kprintf("%ld",i); /* print the value of i */
     thread_fork("helper1",NULL,helper,NULL,i+1); /* fork thread to run helper(NULL,i+1) */
     thread_fork("helper2",NULL,helper,NULL,i+1); /* fork thread to run helper(NULL,i+1) */
  }
  if (i==0) {
     kprintf("%ld",i);
  }
  thread_exit();
}
```

Which of the following outputs could possibly be generated by this program?

[✓ a.] 01221220

[✓ b.] 01120222

c. 01342560

d. 01222120

[✓ e.] 01122220

f. 01234560

[✓ g.] 01212220

[✓ h.] 00112222

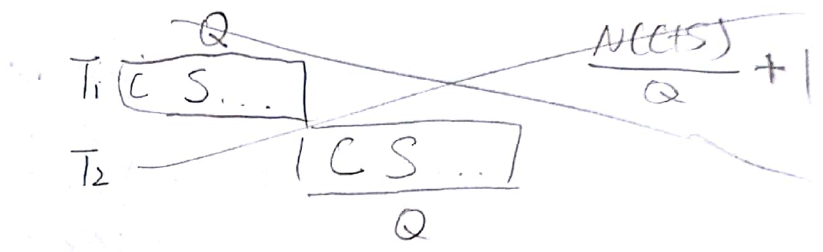
In-Class Problems: twothreads

Suppose that there are two threads in a system that uses preemptive round-robin scheduling with a scheduling quantum of Q milliseconds. The system has a single processor. Each thread runs a function which behaves as follows

```
for i from 1 to $N$ do
  compute for $C$ milliseconds
  sleep for $S$ milliseconds
end
```

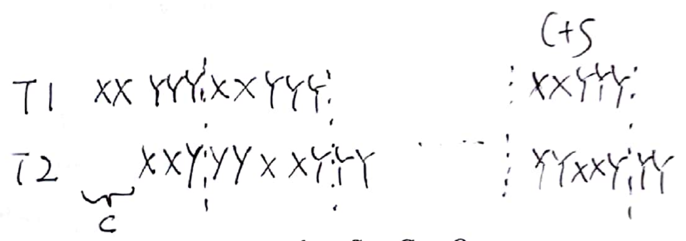
At the end of its for loop, a thread is finished and it exits. During the "compute" part of each iteration, a thread is runnable (running or ready to run). During the "sleep" part of each of its iterations, a thread is blocked. For both parts of this question assume that $C < Q$ and $C < S$.

a. First, assume that $C < S$ and $C < Q$. Suppose that both of the threads are created at time $t = 0$. At what time will both of the threads be finished? Answer in terms of Q, N, C, and S, as necessary.



Compute = XX

Sleep = YYY



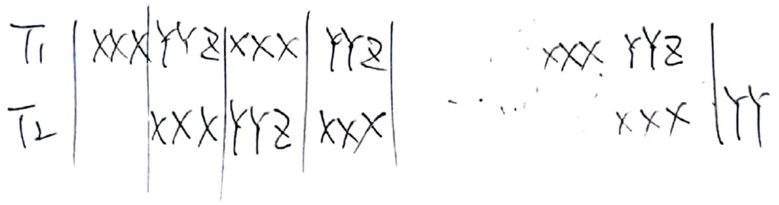
$N(C+S)+C$

b. Answer the same question, but this time assume that $S < C < Q$.

Compute = XXX

Sleep: YY

Z = extra waiting



$2NC+S$

In-Class Problems: OS/161 Locks

Sketch implementations of OS/161 lock_acquire and lock_release, using a spinlock and a wait channel. Also, determine what fields you'll need in the lock structure.

```
struct lock {
        char *lk_name;
        spinlock * sl;
        wchan * wc;
        cpu * held;

};

void lock_acquire(struct lock *){ <- KASSERT(lk != NULL)
                                     KASSERT(!Lock_do_i_own(lk))
critical section
    spinlock_accquire(lk -> sl)
        while (lk->held)  // if the lock is owned by other threads
        {  wchan_lock(lk->wc)
           spinlock_release(lk-sl)
           wchan_sleep(lk -> wc)
           spinlock_acquire(lk->sl)
        }
        lk->held = true
        lk-> owner = curthread
    spinlock_release(lk->sl)
}

void lock_release(struct lock *){ <- KASSERT(lock_do_i_own(lk)) <- MOST OWN this lock
    spinlock_accquire(lk->sl)
    lk -> held = false
    lk -> owner = NULL
    wchan_wakeone(lk->wc)

    spinlock_release(lk->sl)


}
```

In-Class Problems: queuexfer

Suppose that a threaded program has N queues of items. The program needs to support an operation called `Transfer(i,j)`. Each call to `Transfer` will transfer a single item from the ith queue to the jth queue, unless there is nothing in the ith queue, in which case the call will not affect the queues.

The program will have multiple concurrent threads, each of which may call `Transfer` zero or more times.

How would you use locks to ensure that `Transfer` operations are atomic? Specifically, how many locks would you use, what would each lock protect, and when would the locks be acquired and released to ensure that transfers are atomic?

```
① Transfer(i,j)          — Single Thread.
     acquire(lock)
          : work
     release(lock)

② Transfer(p,q) — p-th queue
                  q-th queue   resource ordering, avoid deadlock
    if (p.num < q.num)
        acquire(P->lk)
        acquire(Q->lk)
  else
       acquire (Q->lk)          transfer(Q5,Q1)
      acquire (P->lk)
   :
③ Transfer(P,Q)
     acquire(P)
       tmp = P->pop()
     release(P)
     acquire(Q)
       Q->push(tmp)
     release(Q)
```

In-Class Problems: Semaphores to CVs

Global Variables	Initialization	Function func1	Function func2

`struct semaphore *sa;`
`struct semaphore *sb;`
`struct semaphore *sc;`
	`sa = sem_create("A",1);`
	`sb = sem_create("B",1);`
	`sc = sem_create("C",0);`
		`void func1() {`
		`P(sa);`
		`funcA();`
		`V(sa);`
		`P(sc);`
		`}`
			`void func2() {`
			`P(sb);`
			`funcB();`
			`V(sb);`
			`V(sc);`
			`}`

Re-implement `func1` and `func2`, from the program above, using locks and condition ~~variables~~ instead of semaphores. In addition to locks and condition variables, you can define and use other global variables.

```
void func1() {
    acquire_lock( lock1)
    funcA()
    release _lock(lock1)
    acquire lock3
    while( count  == 0) {

        cv_wait

    }
    count --;
    cv_signal (not_full )
  release  lock3
```

```
void func2()
  acquire_ lock( lock 2)
  funcB()
  release_ lock ( lock 2 )
  acquire lock 3
  while ( count  == N ) {
      cv_wait
  }
  count ++
  cv_ signal (not empty ).
  release lock3 .
```

In-Class Problems: procfork

Question 1: What output will be generated by the parent and child processes for the program shown below?

```
int x;  /* global variable */
int main()
{
  int rc;
  x = 0;
  rc = fork();
  if (rc == 0) {
    x = 10;
    printf("A: %d\n", x);
  } else {
    printf("B: %d\n", x);
    x = 100;
  }
  printf("C: %d\n", x);
}
```

Parent — Child

rc = fork(): //return childpid → rc = fork(): // return 0.

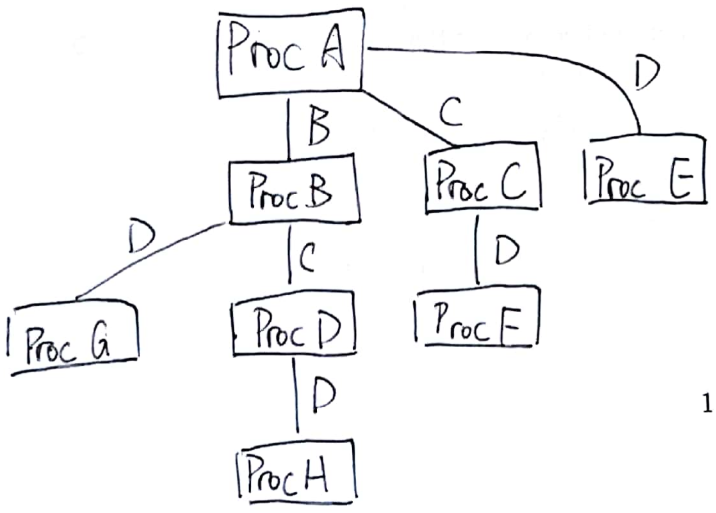
P: B: 0 C: A: 10

C: 100 C: 10

Seperate Address.

Question 2: Consider a system with a process, ProcA, that executes the program below. Draw the tree rooted at ProcA representing the processes created by this program. Every node in the tree should represent a process, and there should be an edge from node A to node B if process A creates process B. Label each node of the tree with the label of the fork() system call that created it.

```
int main () {
  fork();    /* Label: B */
  fork();    /* Label: C */
  fork();    /* Label: D */
  return 0;
}
```



In-Class Problems: Simple Paging

Consider a paging-based virtual memory system with 32-bit virtual and physical addresses, and a page size of 2^{12} bytes (4KB). Suppose that process P is running. P uses only 128KB of virtual memory. The first 5 entries of P's page table are shown below.

Page #	Frame #	Valid
0x0	0x00234	1
0x1	0x00235	1
0x2	0x0023f	1
0x3	0x00ace	1
0x4	0x00004	1

Answer the following questions:

Q1: What is the total number of entries in P's page table?

#Pages $2^{32}/2^{12} = 2^{20}$

#bits for PG# = 20

bits 4 offset # = 12

Q2: How many of the entries are valid?

$\frac{128\,KB}{4KB} = 2^5 = 32$ pages $\left[\frac{ADDRSPCSZ}{PGSZ}\right]$

Q2: Which physical addresses correspond to each of these virtual addresses?

- 0x00001a60 0x235A60
- 0x00000fb5 0x234FB5
- 0x00004664 0x4664

Q3: If the page size were 16KB (2^{14}) instead of 4KB, how many entries would there be in P's page table? How many bits of each virtual address would be used for the offset, and how many for the page number?

$2^{32}/2^{14} = 2^{18}$ pages.

offset : 14

page number = 18

In-Class Problems: Multi-Level Paging

Consider a virtual memory system that uses multi-level paging for address translation. Virtual addresses and physical addresses are 64 bits long. The page size is 1 MB (2^{20} bytes). The size of a page table entry is 16 (2^4) bytes. Each individual page table, at each level, must fit in a single frame.

Q1: How many bits of each virtual address are needed to represent the page offset?

$$\log(\underset{\substack{\uparrow\\ \text{PAGESIZE}}}{2^{20}}) = 20$$

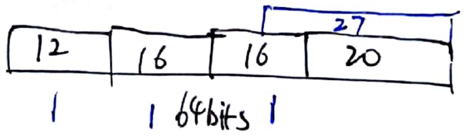
Q2: What is the maximum number of entries in an individual page table?

$$2^{20}/2^{4} = 2^{16} \longrightarrow \#\text{PGBITS} = 16.$$

Q3: What is the number of levels of page tables that will be required for virtual-to-physical translation in this system?

$$\#\text{Levels} = \left\lceil \frac{64-20}{16} \right\rceil = 3$$

Q4: Suppose that a particular process uses only 128 MB (2^{27} bytes) of virtual memory, with a virtual address range from 0 to $2^{27} - 1$. How many individual page tables, *at each level*, will be required to translate this process' address space?



2^7 valid entries

Try 41 bits.

In-Class Problems: Address Translation on the MIPS

Suppose that the TLB in the MIPS MMU contains the following entries. For each entry, only the virtual page number and physical frame number are shown. Assume that all of the entries are valid. The page size on the MIPS is 4KB.

$2^{12} = 1 0000 0000 0000$

$= 0x1000$

entry number	virtual page number	frame number
0	0x10000	0x08343
1	0x10004	0x08344
2	0x00400	0x0100a
3	0x7ffff	0x01112
4	0x7fffe	0x01113
5	0x00402	0x0600b
6	0x00404	0x01114

For each of the following *physical* addresses, indicate which virtual addresses will translate to that physical address. If more than one virtual address translates to a physical address, list them all.

a. (3 marks) 0x01113fa0

User: 0x7fffefa0

kernel0: 0x81113fa0.

kernel1: 0xA1113fa0.

b. (3 marks) 0x0100a088

User: 0x00400088

Kernel0: 0x80400088.

Kernel1: 0xA0400088.

c. (3 marks) 0x0100b014

Kernel0: 0x8100b014

Kernel1: 0xA100b014

d. (3 marks) 0x08343ffc

User: 0x1000 0ffc

Kernel 0: 0x 8834 3ffc

Kernel1: 0xA834 3ffc

In-Class Problems: Disk I/O

Suppose that a server has a single disk drive and one single-core processor. A total of k processes are running in the system. Each process, if it were running alone in the system, would issue a request to retrieve a 4KB (2^{12} bytes) block of data from the disk after every 5 milliseconds of run time on the CPU.

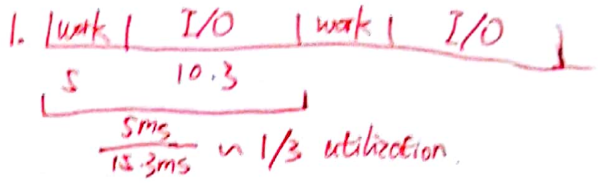
The disk drive has 1024 (2^{10}) tracks and a total capacity of 128 MB (2^{27} bytes). According to the manufacturer, the drive's average seek time is 5 milliseconds, and the disk spins at 100 rotations per second.

Q1: Suppose that $k = 1$. Estimate the CPU utilization, i.e. the fraction of the time that the CPU is not idle.

Steps:

1. Estimate time for I/O request
   a) What is the seek-time? avg 5ms
   b) cost of full rotation? 10ms
   c) avg rotational latency? 5ms
   d) how many bytes per track? $2^{27}/2^{10} = 2^{17}$ bytes
   e) how many blocks per track? $2^{17}/2^{12} = 2^5$
   f) transfer time for 1 block? $\frac{1}{32} \times 10ms \approx 0.3ms$
   g) seek + rotate + transfer =
      5 5 0.3 10.3ms

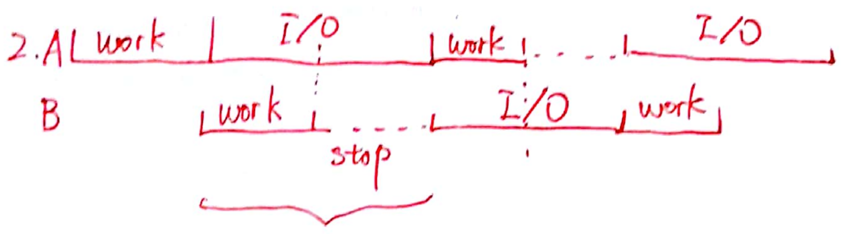
2. now calc cpu idle time



$\frac{5ms}{15.3ms} \sim 1/3$ utilization.

block = 1 sector for this Q.

Q2: Repeat Q1, assuming that $k = 2$.



$\frac{5ms}{10.3ms} \approx \frac{1}{2}$ utilization.

try 3 threads.

CS350 Operating Systems

In-Class Problems: Pathname Translation

Write a pseudo-code procedure to translate a pathname into an i-number in a hierarchical file system. Your procedure should take a pathname (of type string) as its only parameter. It should return the i-number (of type integer) of the file specified by the input pathname. If any error is encountered during translation, your function should return the special code *INVALID* instead of a real i-number. Assume that the i-number of the root directory is 0. To simplify the pseudo-code, treat strings like primitive data types.

Your procedure should use the following functions:

`boolean is_directory(int i-number)` This function returns the value "TRUE" if the file whose i-number is given is a directory. It returns "FALSE" otherwise.

`int num_components(string pathname)` This function returns the number of components in the specified pathname. For example, num_components("*/a/b/c*") returns 3, num_components("*/foo*") returns 1, and num_components("/") returns 0.

`string get_component(int i, string pathname)` This function returns a string representing the *i*th component of the specified pathname. The number "i" should be a positive integer. For example, get_component(*2*, "*/a/b/c*") will return the string "b", and get_component(*1*, "*/foo/bar*") will return "foo". If there is no *i*th component, the function returns the empty string "".

`int dsearch(int i-number, string component)` This function is used to search a directory file for an element whose name matches the "component" argument. The parameter "i-number" must be the i-number of the directory file to be searched. The function returns the i-number of the matching entry, For example, the call dsearch(*10*, "*foo*") will return the i-number of the "foo" entry in the directory file whose i-number is 10. If there is no such entry in the specified directory, the function returns the value -1.

~~counter = 0~~
~~hashmap<string, int> map~~

~~int num-components (string pathname)~~
~~if (map. contains (pathname)) return map[pathname]~~
~~counter++;~~
~~map [counter] = pathname~~
~~return counter;~~

~~boolean is_directory (~~

```
int translate (string path)
{
    int inum = 0
    int i = 1
    for i = 1 to num_components (path)
    {
        if (! is_dir (inum)) return ~~INVALID~~ INVALID
        inum = dsearch (inum, get_comp(path, i))
        if (inum < 0) return INVALID
    }
    return inum
}
```

In-Class Problems: File Systems

Consider the following fragment of an application that accesses a file stored in a VSFS-like file system:

```
char x[1000];
// assume that f is a descriptor referring to a
// large file that has already been opened
lseek(f,10500,SEEK_SET); ← None read, None write.
write(f,x,1000);
read(f,x,1000);
```

Assume that the file system uses a block size of 1024 bytes, and that i-nodes include 9 direct pointers, 1 indirect pointer, and one double-indirect pointer. The file system has a block cache (intially empty) and an i-node cache.

Estimate the number of read and write operations the file system will need to perform to implement each of the file system operations in the application program.

block size = 1 kB

block ptr size = 4 Bytes

ptr PerBlock = 1 kB / 4
= 256

in bytes.

9 direct = 9×1 = 9 kB → [0, 9216)

1 single = 1×256×1kB = 256 kB → [9216, 271360]

1 double = 1×256^2×1kB = 65536 kB → [271360, ...).

max file size = 9 kB + 256 kB + 65536 kB
= 65801 kB

Current Pos: 10500

ind block

→ [9216, 10240)

→ [10240, 11264) — write start

→ [11264, 12288) = write end, read start 11500

→ [12288, 13312) - read end 12500

Write: inode read → put in cache

read single ind. → put in cache

read 2 x data → put in cache (B+C)

write 2 x data B+C

write inode

1 inode R, 1 inode W

3 block R, 2 block W.

Read: 1 block Read(D).

(B, C in cache)

1 inode W