Problem Sheet for B16, Operating Systems, Havoutis, Hilary 2018

0.) Use ssh and your engineering department credentials to remotely login to one of engs-station41(42, 43, 44, ...77).eng.ox.ac.uk machines. Be warned that for some crazy reason it appears that these machines are turned off on the weekend so do these first questions during the week to be safe.

1.) Use “df” to find out what size blocks are used to partition the nonvolatile memory tracked by the filesystem.

2.) Use “man getpagesize” to get information about the Linux glibc OS command such that you can write a 7-line nicely-formatted c program to print-out the page size used by the operating system to partition volatile memory. What is the pagesize used by the installed OS?

3.) Use “free” to figure out what the total volatile storage available on the machine is in gigabytes.

4.) Use “man top” to learn about the top command then use the sort field ordering of memory to list the top 5 memory consuming process names on the system. Note the user responsible for said application.

5.) Use “echo $$” to get the process id of your current bash shell. Use “pmap –x X” where X is the value of your process id to list the pagemap used by the shell program.
a) Is the address space a 64 or 32 bit address space?

b) What is the address of the first byte of machine code? Extra credit: Why?

c) Where is the stack and which direction does it grow?

6.) If the executable you wrote for (2.) is called “a.out” use “strace ./a.out” to count the number of system calls made when “a.out” is executed. Approximately how many are executed?

7.) Recursion can be used to compute the nth Fibonacci number in the following way:

```
int fib(int n) {
        if(n==0)
                return 0;
        if(n==1)
                return 1;
        return fib(n-1) + fib(n-2);
}
```

a) What is the largest value of n that could be computed on a machine with 1MB memory if the OS pushes, including function parameters, 32 bytes to the stack for every function call?

b) Extra credit : What could be done to increase the largest computable value?

8.) Describe an efficient mechanism by which an OS can implement interprocess communication. Describe a distinct, less efficient mechanism too.

9.) Suppose a, b and c are three arrays of integers in a C program. Why, when N is large, might

```
for (j = 0; j < N; j++)
        for (i = 0; i < N; i++)
                c[i][j] = a[i][j] + b[i][j];
```

take a thousand times longer than

```
for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
                c[i][j] = a[i][j] + b[i][j];
```

on the same machine?

10.) Use semaphores to solve the producer consumer problem. Assume the existence of a single reader, single writer, a character array buffer, and two semaphores; one labeled empty, the other labeled occupied. Other variables may be instantiated as needed.

```
Semaphore empty = B;
Semaphore occupied  = 0;
int nextin = 0;
int nextout = 0;
```

```
void Produce(char item) {
   P(empty);
   buf[nextin] = item;
   nextin = nextin + 1;
   if (nextin == B)
      nextin = 0;
   V(occupied);
}
```

```
char Consume( ) {
   char item;
   P(occupied);
   item = buf[nextout];
   nextout = nextout + 1;
   if (nextout == B)
      nextout = 0;
   V(empty);
   return(item);
}
```

11.) On the slide titled “Spin-Lock Implementation of Blocking Mutex In Hyp. OS,” a subtle thread synchronization problem can happen. What is it?

```
void blocking_lock(mutex_t *mut) {
  spin_lock(mut->spinlock);
  if (mut->holder != 0)
    enqueue(mut->wait_queue, CurrentThread);
    spin_unlock(mut->spinlock);
    thread_switch();
  } else {
    mut->holder = CurrentThread;
    spin_unlock(mut->spinlock);
  }
}
```

```
void blocking_unlock(mutex_t *mut) {
  spin_lock(mut->spinlock);
  if (queue_empty(mut->wait_queue)) {
       mut->holder = 0;
  } else {
       mut->holder = dequeue(mut->wait_queue);
       enqueue(RunQueue, mut->holder);
  }
  spin_unlock(mut->spinlock);
}
```

12.) A multithreaded server processes web requests and streams audio data to 100 clients simultaneously.

a) Assuming a 100Mb/sec network interface card what is the maximum achievable audio stream rate to each client assuming a 3% packet header network overhead and perfect sharing of the network and CPU?

b) Describe using diagrams and text the data flow, scheduling, interrupt, and switching behavior of a straight threads OS running on a single core CPU required to serve each of these clients.

c) Note in particular the OS services required and used. Also, compute the order of magnitude of the minimum CPU clock speed required to saturate the network if memory reads and writes take two CPU clock ticks, device I/O is PIO-style, and saving context including register state takes five CPU clock ticks.

13.) Assume 8-Kb pages, how big is a page table for a 64-bit architecture? What are some strategies for dealing with this problem?

14.) Write a TCP/IP-based client server application that runs *nix text commands on a remote host and streams the response back to the client. The client command should have an interface like “client <ipaddr> <textofcmd>” where ipaddr is the ip address of a machine, something like “nslookup engs-station60.eng.ox.ac.uk” -> 163.1.140.60 and <textofcmd> is the command to run on the server, e.g. “/usr/bin/uptime.” Write the server side such that there is a single thread that accepts connections and creates a thread to service each request. This thread should create a pipe, fork, close stdout and stderr, use dup to redirect stdout and stderr to the correct end of the pipe, and in the parent (of the forked thread) read the output from execl’ing the command using a buffer to intermediate between the pipe and the socket connection back to the client. Note that you can use gcc and any of the linux software laboratory machines, e.g. engsstation60. eng.ox.ac.uk to as the server in this task. Note that ports 8889 and above are good ports to use as they have user bind permissions.