CS240 Midterm Answers 07/24/2014

P1(a) 12 pts

The character 'B" is printed.
8 pts

No.
4 pts

P1(b) 12 pts

x contains an address of a location that stores a value of type int. The address that x contains has not been allocated with memory. Hence it is likely to be an address that the process running the code snippet is not allowed to access. This triggers a segmentation fault.
6 pts

Before the statement *x = 5 add the statement
x = (int *) malloc(sizeof(int));
6 pts

P1(c) 12 pts

A new process is created that runs the same code as the process (parent) that created it (child).
4 pts

fork() returns to the parent process with the PID value of the child process. fork() returns to the child process with value 0.
4 pts

In a concurrent server, after creating a new process using fork(), the parent goes back to handling new client requests while the child carries out the task requested by a client. For this to happen, the parent process needs to know that it's the parent and the child process that it's the child. The return values of fork() make it easy to differentiate.
4 pts

P2(a) 16 pts

A mask that is all 0's except in the least significant bit (bit position 0) is created. This is done by shifting a word containing all 1s to the left by 1 position and then performing a 1's complement operation. x is shifted to the right by 31 positions which relocates the value of the most significant bit to position 0. This shifted x is AND'ed with the mask which blanks out all bits but for the one at position 0.
8 pts

```
m = ~(~0 << 1);           // mask
y = x >> 31;              // shifted x
z = y & m;                // AND mask with shifted x
printf("%u",z);           // print z
```

In the above, m, y, z are defined as unsigned int.
8 pts

P2(b) 16 pts

((B+1)+2))
8 pts

As discussed in class, 2-D arrays use double indirection. B+1 adds 1 to the address stored in B. This address, in turn, contains an address: namely, the starting address of a contiguous area of memory where the elements of B[1][0], B[1][1], B[1][2] are stored. *(B+1) accesses this starting address, adding 2 specifies the address of the third element B[1][2]. Finally, applying * accesses the content of this address, that is, the value stored in B[1][2].
8 pts

P3(a) 16 pts

An interative server takes a client request, parses it, and carries out the requested task itself. A concurrent server delegates the task requested to a child process and goes back to waiting for the next client request.
3 pts

The client request to a shell may be to run a binary such as /bin/ls. Performing execl() in an iterative server will overwrite the server binary with /bin/ls, thus destroying it. Thus it cannot go back to prompting for future requests which a shell must be able to do.
3 pts

```
while(1) {

// read input from client
// parse input from client

// delegate task to child process
  k = fork();
  if (k == 0) {   // child code
  // perform requested client task
  // such as execl() to execute a binary
  }
  else {          // parent code
  // wait for child to complete or
  // immediately go back to reading next
  // client request
  }
}
```
4 pts

In the parent code section (the "else" part).
2 pts

It allows the child process to complete its processing which may entail writing to stdout. Only then does the parent go back to the beginning of the loop, printing a prompt. Hence the prompt character(s) does not get interleaved with output generated by the child process.
2 pts

When a user requests that a process be run in the background, for example,

```
% /bin/ls &
```

then the shell should immediately return a prompt waiting for the
next user request. Calling waitpid() in the parent section of the
concurrent server code would not allow that.
2 pts

P3(b) 16 pts

```
double *Z;
FILE fp;
int i;

fp = fopen("mill.dat","r");

Z = (double *) malloc(1000000*sizeof(double));
for (i=0; i<1000000; i++)
  fscanf(fp,"%lf",&Z[i]);
```

Bonus

printf() is often used to localize a segment of code where a run-time
error might be occurring. A call to printf() may have completed
and returned to the caller. However, since the stdio library uses buffers
where requested print data are temporarily stored, when a process
terminates due to a run-time error right after printf(), there is no
guarantee that stdio's buffer content will be flushed to the terminal.
fflush(stdout) forces a flush so that finding the code segment
that trigger a bug is made easier.