Advanced Unix programming – Quiz1

Time limit: 12 minutes

1. (1 point) Which of the following is *not* a useful feature of emacs, for C programmers:

   (a) It can indent the program automatically.

   (b) The etags facility.

   (c) It can detect programs that can go into an infinite loop.

   (d) You can compile and go to errors within emacs.

   C.

2. (1 point) Which of the following options should be provided to *gcc*, if you wish to debug the program with a debugger?

   (a) -c

   (b) -g

   (c) -k

   (d) -l

   B.

3. (1 points) Write a macro called 'junk' that takes two parameters, x, and y. If $x < y$, it should give you the value of *x*y*. Otherwise, it should give you the value of *x/y*. Use a conditional *expression* in the macro definition to accomplish this. Make sure that you avoid errors caused by operator precedence.

   ```
   #define junk(x,y) ( ((x)>(y)) ? ((x)*(y)) : ((x)/(y)) )
   ```

4. (2 points) Add guards to the following header file, *myheader.h*, to prevent its contents from being included twice.

   ```
   #ifndef _MYHEADER_H (or anything else)
   #define _MYHEADER_H

   static const double PI=3.1415927;

   #endif
   ```

5. (2 points) Write a single C statement so that a character array *buffer* will contain the string *"file.n.m"*, where n = process ID, and m = user ID. For example, if the process ID is 8631, and user ID is 9876, then *buffer* should contain *"file.8631.9876*. You may assume that the array is large enough to store this string. You need not show the inclusion of any header files.

   ```
   sprintf(buffer, ‘‘file.%d.%d’’, (int) getpid(), (int) getuid);
   ```

6. (1 points) Is the machine *linprog* little-endian or big-endian? If you do not know the answer, you may, alternatively, write a small program that will determine if the machine is little-endian or big-endian.

   Little endian.

7. (2 points) Consider the following situation: You run a program owned by your friend, for which the *set-user-ID* bit is on. (We assume that you have appropriate permissions to run the program.) Which of the following statements is true.

   (a) The real User ID = friend's ID, the effective ID = your ID, and the saved set ID = your ID.

   (b) The real User ID = your ID, the effective ID = your ID, and the saved set ID = friend's ID.

   (c) The real User ID = your ID, the effective ID = friend's ID, and the saved set ID = your ID.

   (d) The real User ID = your ID, the effective ID = friend's ID, and the saved set ID = friend's ID.

   D. (1 point if answer is C.)

Advanced Unix programming – Quiz2

Time limit: 12 minutes

1. (1 point) Where is information about a file, such as the locations of the blocks, actually stored:

   (a) File descriptor.

   (b) i-node.

   (c) Open file description.

   (d) File table.

   B

2. (2 point) Write a short piece of code that redirects standard input from a file called "*in.file*", using *open* and *close*. (You may wish to use the following flags: $STDIN\_FILENO$, and O_RDONLY.)

   ```
   close(STDIN_FILNO);
   open(``in.file'', O_RDONLY);
   ```

3. (1 point) Assume that standard input is redirected from a file called: "*in.file*", while standard output remains the terminal. Which of the following buffering would meet ANSI C requirements:

   (a) Standard input is fully buffered, while standard output is line buffered.

   (b) Standard input is line buffered, while standard output is unbuffered.

   (c) Standard input is unbuffered, while standard output is line buffered.

   (d) Standard input and standard output are fully buffered.

   A

4. (1 point) **Normally**, which type of buffering is used for standard error.

   (a) Line buffered.

   (b) Unbuffered.

   (c) Fully buffered.

   B

5. (2 points) Consider the following program fragment:

   ```
   int f1, f2;
   f1 = open([suitable arguments to open a file for reading and writing]);
   f2 = f1;
   close(f2);
   ```

   Which of the following is true after the "*close*" statement?

   (a) The file descriptor *f1* can be used for reading and writing, but *f2* cannot be used.

   (b) Both *f1* and *f2* can be used for reading and writing.

   (c) Neither *f1* nor *f2* can be used for reading or writing.

   (d) *f1* can be used for reading and *f2* for writing.

C

6. (2 points) Which of the following is true about the order of execution of parent and child processes?

   (a) After calling "fork", the parent waits for the child to complete, before it executes any instruction.

   (b) After the parent calls "fork", the child waits until the parent terminates, before it executes any instruction.

   (c) They can execute instructions in any order.

   C

7. (1 points) Which of the following is true after an "exec"

   (a) File descriptors are left open across an exec, unless the $FD\_CLOEXEC$ flag is set for that descriptor.

   (b) File descriptors are closed across an exec, unless the $FD\_OPENEXEC$ flag is set for that descriptor.

   (c) File descriptors are always closed.

   (d) File descriptors are always left open.

   A

8. (1 point) Which of the following is true about a pipe?

   (a) It can be used for communication between a parent and child.

   (b) It can be used for communication between any two processes.

   (c) It can be used for communication *only* between siblings, but not between parent and child.

   (d) It cannot be used for inter-process communication, but only for communication between different threads of the same process.

   A
   (I gave a point even if your answer was B, but please read section 14.2.)

Advanced Unix programming – Quiz3 Solutions

1. (1 point) What should the data type of *fds* be, for it to be used in the function call *pipe(fds)*? (**Give a correct declaration of *fds***).

   ```
   int fds[2];
   ```

2. (2 points) Consider the function call: *pipe(fds)*. Which of the following **must** be true.

   (a) fds[0] and fds[1] are available for reading.

   (b) fds[0] is available for reading and fds[1] is available for writing.

   (c) fds[0] is available for writing and fds[1] is available for reading.

   (d) fds[0] and fds[1] are available for writing.

   b

3. (1 point) Consider the following situation. A process sets a function called *sig_int* to be the signal handler for SIG_INT. It then execs a program. Will *sig_int* remain the signal handler for SIG_INT in this exec-ed program?

   (a) Yes.

   (b) No.

   (c) All of the above.

   b

   In the rest of the quiz, you may wish to use some of the following functions, data types, and macros (you may need to use a few outside this list too): sigemptyset, sigfillset, sigaddset, sigdelset, sigaction, sigprocmask, sigset_t – *used to store a signal set*, struct sigaction – *which has fields: sa_handler, sa_mask, sa_flags, and sa_sigaction*, SIG_BLOCK, SIG_UNBLOCK, SIG_SETMASK, SIG_DFL, SIG_IGN.

4. (1 points) Write a short piece of code to make the SIG_INT signal ignored. (You may use the function: 'signal', if you wish to.)

   ```
   signal(SIGINT,SIG_IGN);
   ```

5. (4 points) Write a short piece of code to add SIG_INT to the set of signals that are currently blocked. Make sure that you declare correct data types and set up the signal set correctly.

   ```
   sigset_t set;

   sigemptyset(&set);
   sigaddset(&set,SIGINT);
   sigprocmask(SIG_BLOCK, &set, NULL);
   ```

6. (1 point) Write a short piece of code to have SIG_INT handled by a function called sig_int. No additional signal will be blocked during execution of the signal handler, and no special flags will be set. Make sure that you declare correct data types. You may **not** use the 'signal' function.

```
struct sigaction act;

act.sa_handler = sig_int;
sigemptyset(&act.sa_mask);
act.sa_flags = 0;

sigaction(SIGINT, &act, NULL);
```

Advanced Unix programming – Quiz4

Time limit: 12 minutes

1. (1 point) Which of the following statements is true?

   (a) TCP is more reliable than UDP, and UDP is connection-oriented.

   (b) UDP is more reliable than TCP, and UDP is connection-oriented.

   (c) TCP is more reliable than UDP, and TCP is connection-oriented.

   (d) UDP is more reliable than TCP, and TCP is connection-oriented.

   C

2. (4 points) Complete the code fragment given below, so that the *sockaddr_in* structure has the correct IP address (from the variable *IPAddress*) and port number (from the variable *Port*).

```
char IPAddress[]=‘‘128.186.120.33’’;
char Port[] = ‘‘50002’’;
struct sockaddr_in addr;
memset(&addr, 0, sizeof(struct sockaddr_in));
addr.sin_family = AF_INET;

addr.sin_port = htons(atoi(Port));
inet_pton(AF_INET, IPAddress, &addr.sin_addr);
```

3. (5 point) What is the *typical* sequence of calls to the following functions on the client and server, respectively, as discussed in class: (i) connect, (ii) socket, (iii) bind, (iv) close, (v) reads/writes, (vi) accept, (vii) listen.

```
Server                          Client
------                          ------

socket                          socket
bind                            connect
listen                            reads/writes
accept                          close
  reads/writes
close
```

Advanced Unix programming – Quiz5

Time limit: 12 minutes

1. (1 point) Which of the following functions is used for setting socket options?

   (a) setsockopt

   (b) putsockopt

   (c) inputsockopt

   (d) writesockopt

   A.

2. (6 points) Assume that a client needs to read on two different file descriptors, (i) standard input and (ii) a socket. When data from standard input is available, it will call a function: *R1(sockfd)*. When data is available from the socket, it will call a function *R2(sockfd)*. Write code in the *for* loop below, so that the client can use *select* to respond to whichever file descriptor is ready for reading. You need not handle errors, or close the socket on *EOF*.

```
int sockfd, maxf;
fd_set  rset;
 ... Some code ...
  for(;;)
  {
    FD_ZERO(&rset);
    FD_SET(STDIN_FILENO, &rset);
    FD_SET(sockfd, &rset);
    maxf = sockfd + 1;

    select(maxf, &rset, NULL, NULL, NULL);

    if(FD_ISSET(sockfd, &rset))
      R2(sockfd);
    if(FD_ISSET(STDIN_FILENO, &rset))
      R1(sockfd);
  }
```

3. (1 point) In the pre-forked server example that we discussed in class, which of the following is true:

   (a) Each child listens on a different socket.

   (b) All the children listen on the same socket.

   B.

4. (2 point) In the pre-forked server example that we discussed in class, which of the following is true on a Linux system, after the first client connection arrives on the completed connection queue:

   (a) *accept* returns in each child, with an identical file descriptor number.

   (b) *accept* returns in each child, with different file descriptor numbers.

   (c) *accept* returns a file descriptor in one of the children; the other children are put back to sleep.

   (d) *accept* returns a file descriptor in one of the children; the other children are terminated.

   C.

Advanced Unix programming – Quiz6

Time limit: 12 minutes

1. (1 point) Which scheme did we use in class to impose flow and error control in a UDP application?

   (a) Check sum
   (b) Sliding window protocol
   (c) Thundering herd
   (d) Positive acknowledgment with retransmission

   D.

2. (1 points) Which of the following is **not** mentioned in the text as a possible reason for retransmission?

   (a) The request (original message) is lost
   (b) The client received an RST
   (c) The reply (acknowledgment) is lost
   (d) The retransmission timeout is too small

   B.

3. (2 points) Which of the following is **not** a feature **we** added, to make the UDP client/server reliable?

   (a) Handshake
   (b) Timeout and retransmission
   (c) Checksum
   (d) Sequence number

   C.

4. (1 point) Which of the following is used to create a POSIX thread?

   (a) pthread_connect
   (b) pthread_create
   (c) pthread_socket
   (d) pthread_accept

   B.

5. (1 point) In which of the following cases can a Unix domain protocol be used?

   (a) Client and server are on the same host, running Linux.
   (b) Client and server are on two different hosts, both running Solaris.
   (c) Client is on a host running Solaris, and server is on a host running Linux.
   (d) Client is on a host running Linux, and server is on a host running Solaris.

   A.

6. (1 point) Give arguments to the socket system call to create a (datagram or byte stream) Unix domain socket. Your answer may conform either to the POSIX specification, or with the example discussed in class.

```
socket(AF_UNIX /* or AF_LOCAL */, SOCK_STREAM, 0 );
```

7. (3 points) Declare a variable that can store the address information of a unix domain socket, and use it to bind a socket descriptor, say sockfd, to */tmp/junk*.

```
struct sockaddr_un addr;

memset(&addr, 0, sizeof(addr)); /* or set a '\0' later */
addr.sun_family = AF_UNIX; /* or AF_LOCAL */
strncpy(addr.sun_path, ``/tmp/junk'', sizeof(addr.sun_path)-1);
bind(sockfd, (struct sockaddr *) &addr, sizeof(addr));
```

Advanced Unix programming – Quiz7

Time limit: 12 minutes

1. (1 point) Which of the following functions is used to wait for a thread to terminate?

   (a) pthread_detach
   (b) pthread_exit
   (c) pthread_join
   (d) pthread_self

   C.

2. (1 point) Which of the following functions is used by a thread to obtain its ID?

   (a) pthread_detach
   (b) pthread_exit
   (c) pthread_join
   (d) pthread_self

   D.

3. (1 point) Which of the following functions is used by a thread to terminate itself?

   (a) pthread_detach
   (b) pthread_exit
   (c) pthread_join
   (d) pthread_self

   B.

4. (1 point) If a thread knows that no thread cares about its return value, then which of the following functions does it call to indicate that resources associated with it can be released on termination?

   (a) pthread_detach
   (b) pthread_exit
   (c) pthread_join
   (d) pthread_self

   A.

5. (1 points) Which of the following is shared by all threads in a process?

   (a) Stack
   (b) Program counter
   (c) File descriptors
   (d) errno

   C.

6. (2 points) Write a code fragment that creates a thread that executes a function called f. The NULL pointer is passed as an argument to this function.

```
pthread_t tid;

pthread_create(&tid, NULL, f, NULL);
```

7. (3 points) Declare and initialize an object that can be used for mutual exclusion. Then use it with *pthreads* functions that support mutual exclusion, so that the following statement can be executed only by one thread at a time.

```
pthread_mutex_t ME = PTHREAD_MUTEX_INITIALIZER; /* Or call
  pthread_mutex_init(&ME,NULL) instead of static initialization */

pthread_mutex_lock(&ME);
counter++;
pthread_mutex_unlock(&ME);
```

Advanced Unix programming – Quiz8

Time limit: 12 minutes

1. (2 points) Let us assume that we want a function, say *void f(void)*, to be called by only one thread. Which *pthreads* function is used to ensure this?

   `pthread_once`

2. (4 points) Declare a variable that can be used to store the "key" for thread-specific data, and then write a statement that will create a key, and set the "destructor" for that key to a function, f.

   ```
   pthread_key_t key;

   pthread_key_create(&key, f);
   ```

3. (2 points) Which *pthreads* function is used by each thread to associate a particular memory location with a particular key?

   `pthread_setspecific`

4. (2 points) Which *pthreads* function is used by each thread to fetch the memory location associated with a particular key?

   `pthread_getspecific`

Advanced Unix programming – Quiz9

Time limit: 12 minutes

1. (6 points) We want a thread executing the function f to wait (without polling) until $rn > 1.0$, before subtracting *1.0* from *rn*. You can assume that some one will signal a conditional variable *cond* whenever *rn* becomes larger than 1.0. Please insert code below to ensure the desired behavior. Please remember to declare and initialize the conditional variable too.

```
static double rn=0.5;
static pthread_mutex_t rn_lock = PTHREAD_MUTEX_INITIALIZER;
(static) pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *f(void *arg)
{
  pthread_mutex_lock(&rn_lock);

  while(rn <= 1.0)
     pthread_cond_wait(&cond, &rn_lock);

  rn -= 1.0;

  pthread_mutex_unlock(&rn_lock);

  return NULL;
}
```

2. (2 points) Which function is used to signal a conditional variable:

   (a) pthread_signal
   (b) signal
   (c) sigaction
   (d) pthread_cond_signal

   D.

3. (2 points) Assume that a thread is blocked on a call to *pthread_cond_wait*. Which of the following is true.

   (a) It releases the associated mutex lock when it is blocked, and re-acquires it when *pthread_cond_wait* returns.
   (b) It releases the associated mutex lock when it is blocked, and does *not* re-acquire it when *pthread_cond_wait* returns.
   (c) It holds the associated mutex lock for the entire duration that it is blocked, and continues to hold it after *pthread_cond_wait* returns.
   (d) It holds the associated mutex lock for the entire duration that it is blocked, but releases it when *pthread_cond_wait* returns.

   A.

Advanced Unix programming – Quiz 10

Time limit: 10 minutes

1. (3 points) For each of the following entities, mention whether it is *shared* by all threads of a process, or if each thread has a *distinct* one.

   - Signal mask – Distinct
   - Signal disposition – Shared
   - Signal handlers – Shared

2. (1 points) Can a thread **ignore** a signal, while other threads in that process do not ignore it?

   No.

3. (1 points) Can a thread **block** a signal, while other threads in that process do not block it?

   Yes.

4. (1 point) Which of the following is normally the default terminal I/O mode?

   (a) Canonical mode
   (b) Non-Canonical mode

   A.

5. (2 points) Which of the following is the main feature of signal driven IO?

   (a) The kernel sends us a signal when something happens on a file descriptor.
   (b) The kernel calls *pthread_cond_signal* when something happens on a file descriptor.
   (c) I/O operations return EWOULDBLOCK if they cannot complete immediately.

   A.

6. (2 points) Which of the following is the main feature of non-blocking I/O?

   (a) The kernel sends us a signal when something happens on a file descriptor.
   (b) Typical input and output calls, and *accept*, return EWOULDBLOCK if they cannot complete immediately, while *connect* returns EINPROGRESS if it cannot complete immediately.
   (c) The kernel terminates the process if an I/O operation cannot complete immediately.
   (d) The kernel sends the process a SIGIO signal, if an I/O operation cannot complete immediately.

   B.