COMP204 - Systems Programming (Sep. 2000)

Time allowed: Two hours

Instructions to candidates:

Answer **eight** questions from Section A and **two** questions from Section B.

SECTION A

Give brief and relevant answers to any **eight** of the following ten questions:
[5 marks each]

(1) Explain the differences between UNIX system calls and library routines, giving one example of each.

(2) Describe two ways of passing back a termination status from a C program, and distinguish between them.

(3) With the aid of examples, describe the differences between low-level and high-level input-output routines.

(4) Why do many UNIX commands read their data from the standard input stream when an input file is not specified?

(5) What is meant by UNIX 'special' files, and where are these kept on the system?

(6) What is an *inode*, and what does it contain?

(7) Explain what is meant by a *zombie* process.

(8) What is meant by a *regular expression*? Give three examples of UNIX tools which make use of regular expressions.

(9) Distinguish between the syntax analysis and semantic analysis phases of program compilation.

(10) Give three examples of faults that might cause a process signal to be generated on a UNIX system.

SECTION B

This section calls for fuller solutions than those of the previous section.

Answer any **two** of the following three questions:

[30 marks each]

B1

(a) Draw a diagram to illustrate the effects in memory of making the following declarations:

```
char colour1[] = “green”;
char colour2[] = “white”;
char *col1 = colour1;
char *col2 = colour2;
```

Now explain carefully the effects of the following:

```
(i)  *col1++ = *col2++ ;
(ii) *col1++ = (*col2)++ ;
```

[8 marks]

(b) Use the above to show how you could write a string copy routine in C.

[8 marks]

(c) What is the problem with the following?

```
char *name;
scanf(“%s”, name);
```

[4 marks]

(d) In the following, the programmer has acknowledged the above problem and has attempted to rectify it:

```
char *func(void)
{ char vec[100];
  return vec;
}

main()
{ char *name;
  name = func();
  scanf(“%s”, name);
  .
  .
  .
}
```

What did the programmer have in mind, and what was wrong with this way of thinking? Describe in detail what the programmer should have done.

[10 marks]

B2

(a) In packaging up a software system the programmer usually creates a README file. What sort of information should this contain?

[5 marks]

(b) Below is a makefile that might typically be associated with such a packaged system:

```
OBJS1 = first.o second.o
OBJS = $(OBJS1) third.o
LIBS = -lm

finalprog: $(OBJS)
      cc $(OBJS) $(LIBS) –o finalprog

$(OBJS1): defs.h

tidy:
      rm –f $(OBJS)

setup:
      mv finalprog /usr/bin/finalprog
```

Explain in detail the purpose and effects of each of the following calls to the given makefile:

```
(i)    make
(ii)   make CFLAGS=-Aa LIBS=-ly
(iii)  make tidy
(iv)   make setup
```

[12 marks]

(c) Suppose make has been called once, as in (i) above. What would happen if we now altered the file defs.h and then called it again?

[6 marks]

(d) Such packages are often bundled up as an archive file. Typically, what UNIX tools are available for creating and unpacking archives, and how are these tools used?

[7 marks]

B3

(a) Say what the following command line will do, and explain your answer:

```
$ who | wc –l >outfile
```

[5 marks]

(b) Now describe how you would set about developing a C program to implement the effects of the above command line. Do not aim to produce a working program; it is far more important that you go into some detail describing each of the steps required to create and manipulate the processes and resources. Your solution should not rely on calls to a shell or any other form of command interpreter to act on its behalf.

[20 marks]

(c) What would be the effect of adding an ampersand (‘&’) to the end of the command line shown in part (a), and what effect would that have on your solution for part (b)?

[5 marks]