A. (8 marks)

Give the appropriate expansion for each of the following acronyms or contractions:

1. *UUCP*

2. *NIS*

3. *IP*

4. *i-list*

5. *PID*

6. *ioctl*

7. *UID*

8. *SIG_IGN*

B. (5 marks)

Consider the following files dealing with system management, administration, or configuration in a BSD UNIX system:

a. `/usr/include/ufs/ufs/dir.h`
b. `/etc/passwd`
c. `/dev/MAKEDEV`
d. `/etc/hosts`
e. `/usr/share/misc/magic`
f. `/usr/include/ufs/ffs/fs.h`
g. `/etc/disktab`
h. `/etc/fstab`
i. `/etc/protocols`
j. `/etc/inetd.conf`
k. `/etc/nsswitch.conf`
l. `/usr/include/ufs/ufs/dinode.h`
m. `/usr/include/stdio.h`
n. `/etc/timezone`

Below are excerpts from certain of the files above. For each of the excerpts, indicate which of the files

(above) it comes from; i.e. for each of the file samples below, give the label of the file above from which the sample was taken.

1. _____

```
#define BBSIZE          8192
#define SBSIZE          8192
#define BBOFF           ((off_t)(0))
#define SBOFF           ((off_t)(BBOFF + BBSIZE))
#define BBLOCK          ((daddr_t)(0))
#define SBLOCK          ((daddr_t)(BBLOCK + BBSIZE / DEV_BSIZE))
 .
 .
 .
#define FS_MAGIC        0x011954
                          /* the fast filesystem magic number */
#define FS_OKAY         0x7c269d38      /* superblock checksum */
 .
 .
 .
#define fsbtodb(fs, b)  ((b) << (fs)->fs_fsbtodb)
#define dbtofsb(fs, b)  ((b) >> (fs)->fs_fsbtodb)
```

2. _____

```
#!/bin/sh -
#
 .
 .
 .
    mknod console           c 0 0
    mknod kmem              c 2 1   ; chmod 640 kmem ; chgrp kmem kmem
    mknod mem               c 2 0   ; chmod 640 mem ; chgrp kmem mem
    mknod null              c 2 2   ; chmod 666 null
    mknod zero              c 2 12  ; chmod 666 zero
    mknod tty               c 1 0   ; chmod 666 tty
    mknod stdin             c 22 0  ; chmod 666 stdin
    mknod stdout            c 22 1  ; chmod 666 stdout
    mknod stderr            c 22 2  ; chmod 666 stderr
```

3. _____

```
ip      0  IP       # internet protocol, pseudo protocol number
icmp    1  ICMP     # internet control message protocol
igmp    2  IGMP     # internet group management protocol
ggp     3  GGP      # gateway-gateway protocol
ipencap 4  IP-ENCAP # IP encapsulated in IP (officially ``IP'')
st      5  ST       # ST datagram mode
tcp     6  TCP      # transmission control protocol
egp     8  EGP      # exterior gateway protocol
pup     12 PUP      # PARC universal packet protocol
udp     17 UDP      # user datagram protocol
hmp     20 HMP   # host monitoring protocol
```

4. _____

```
ftp      stream  tcp   nowait  root    /usr/libexec/ftpd       ftpd -ll
telnet   stream  tcp   nowait  root    /usr/libexec/telnetd    telnetd
shell    stream  tcp   nowait  root    /usr/libexec/rshd       rshd -l
login    stream  tcp   nowait  root    /usr/libexec/rlogind    rlogind -L
finger   stream  tcp   nowait  nobody  /usr/libexec/fingerd    fingerd
#tftp    dgram   udp   wait    root    /usr/libexec/tftpd      tftpd -l
```

5. _____

```
#define IFMT            0170000         /* Mask of file type. */
#define IFIFO           0010000         /* Named pipe (fifo). */
#define IFCHR           0020000         /* Character device. */
#define IFDIR           0040000         /* Directory file. */
#define IFBLK           0060000         /* Block device. */
#define IFREG           0100000         /* Regular file. */
#define IFLNK           0120000         /* Symbolic link. */
#define IFSOCK          0140000         /* UNIX domain socket. */
#define IFWHT           0160000         /* Whiteout. */
```

C. *(1+1+1+2+2+3+4 = 14 marks)*

The following questions require very short, precise answers.

1. In what network did the TCP/IP protocols (and protocol architecture) first appear? I.e. in what network was it first implemented?

2. In UNIX, as with any complex system, there are often synonyms for entities or constructs. What is the synonym for "file"?

3. A program is designed so that it performs the following sequential steps: it uses *gettimeofday(2)* to fill in a `timeval` structure (named `start_time`), performs a long calculation, uses *gettimeofday(2)* to fill in a second `timeval` structure (named `end_time`), and finally uses the *timersub* macro (defined in `<sys/time.h>`) to compute the difference between the `timeval` values in `end_time` and `start_time`. Is the resulting `timeval` difference elapsed time or CPU time?

4. Name two operating systems which are descendants of 4.2BSD UNIX. I.e. name two versions of UNIX that are descendants of 4.2BSD.

5. Name two TCP/IP protocols that are categorized as being (mostly) in the ISO OSI transport layer.

6. Give 3 examples of IPC facilities in a BSD UNIX system.

7. To execute a command, a shell performs a *fork()* call to create a child process, and then the child performs an *exec()* call to start the execution of the desired program. Name four properties or attributes of a process which are inherited across the *exec()* call. I.e. name four process attributes which will stay constant between when the child process calls *exec()* and when the specified program begins execution.

D. (2 marks each = 10 marks)

Answer each of the following questions with a short, precise answer.

1. What directory does the command

```
ls -l ../.
```

list the contents of? (Relate your answer to the current working directory.)

2. It is possible for a file to have permissions set such that its owner cannot read it, even though other users can. Give an example of a file permission pattern (in the format exhibited by "`ls -l`") which would be an example of such a situation.

3. It is possible for *file(1)* to operate without having to read (to use) the contents of `/etc/magic` or `/usr/share/misc/magic` when determining the type of a file. Describe a situation where this would be the case. I.e. give a file type which would be determined by *file(1)* without using `/etc/magic` or `/usr/share/misc/magic`.

4. Convert the IP address whose hexadecimal representation is C22F1582 to "dotted decimal" notation. Give that representation below.

5. What is special about the network address 192.127.127.255?
Hint: what host does this address specify?

E. (6 marks)

Name three built-in (predefined) shell variables (i.e. not user-created variables or environment variables), and say what they are used for (what their purpose or role is). You may choose your examples from either Bourne Shell (*sh*) or C-Shell (*csh*).

F. (8+8=16 marks)

Consider a NetBSD system running on an Intel CPU-based computer. Assume that the system has one physical disk which is partitioned. Suppose that the disk label and partition information for that physical disk, obtained using the *disklabel* command, is as follows:

```
# /dev/rwd0d:
type: unknown
disk: mydisk
label:
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 15
sectors/cylinder: 945
cylinders: 8940
total sectors: 8448300
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

8 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:   305172       63    4.2BSD     1024  8192    16   # (Cyl.    0*- 322)
  b:   526365   305235      swap                         # (Cyl.  323 - 879)
  c:  8448237       63    unused        0     0         # (Cyl.    0*- 8939)
```

```
  d:  8448300        0     unused        0     0          # (Cyl.    0 - 8939)
  e:  7616700   831600     4.2BSD     1024  8192    16    # (Cyl.  880 - 8939)
```

(The "offset" values are in units of physical blocks, as are the "size" values.)

Further suppose that the output of the *mount(8)* command on the same system is as follows:

```
# mount
/dev/wd0a on / type ufs (local)
/dev/wd0e on /var type ufs (local)
/dev/wd0g on /usr type ufs (local)
/dev/wd0f on /exports type ufs (NFS exported, local)
/dev/wd0h on /usr/src type ufs (NFS exported, local)
barbie.usask.ca:/home on /usr/home type nfs
cs:/var/spool/mail on /var/spool/mail type nfs
skorpio3:/faculty/kusalik on /usr/home/kusalik type nfs
```

I.e. the file systems mentioned in the output above are currently mounted, and part of the overall file system.

Based on the information above, give diagrams to satisfy the following specifications.

1. Draw a diagram illustrating the partitioning of the physical disk; i.e. show how the disk is partitioned. In your diagram indicate the label (name) for each partition, where partitions start and end (in units of sectors), what size they have (again in units of sectors), and any partition overlap(s).

2. Draw a diagram — it should have the form of a tree — which describes the overall file system structure on this UNIX system. Be sure to indicate individual component file systems (which have been mounted) and mount points. Also identify in your diagram where each constituent file system originates.

G. (3+4+3 = 10 marks)

Each of the following questions involves a systems programming or administration scenario. Answer each with a short, precise answer based on your systems programming knowledge and experience.

1. A young system manager notices that the (BSD-flavoured) UNIX operating system she has been hired to administer is reporting "hard" errors when accessing certain physical block addresses on a disk partition. The errors come only when writing to or reading from files, but not when file system data structures (such as superblocks, i-list, and cylinder headers) are examined (with `fsdb`, for example). The system manager (rightly) assumes that the disk blocks causing the errors are in the "data space" of the file system; the (physical) blocks for which errors are being reported are being used as data or fragment blocks for files.

After a number of days of monitoring the system, the system manager has compiled a list of all the (physical) block numbers causing the problem on the partition. There turn out to be nine blocks for which errors are being reported. In examination of the file system with *fsck*, the system manager also determines that — luckily — all the blocks causing the problems are currently not allocated to any files.

The system manager decides to eliminate the occurrence of the errors with the following idea: allocating the error-causing blocks to a file, and then never again reading that file nor removing it. Thus, the system manager wants to create a file whose data blocks contain the physical blocks causing the errors. Then by never accessing the contents of the file (e.g. by setting the permissions on the file to allow no access to anyone, and by having all users with root-access privileges refrain from accessing the file) the physical blocks causing the errors will never be read or written.

To implement this idea the system manager intends to write a program which will

- create and open the file (to be used to hold the "bad" blocks) with *open(2)* or *fopen(3)*;
- based on the "bad" physical block addresses collected, *lseek()* to the given block number and write data into each "bad" block using *write(2)* or *fwrite(3)*, thus placing each block in the file;
- close the file with *close(2)* or *fclose(3)*.

The system manager expects the *write(2)* (or *fwrite(3)*) operations will cause the operating system to generate error messages because the disk will report "hard errors". She feels this is acceptable since no data is meant to be stored. All that is necessary is that the "bad" block be allocated as belonging to the file.

The young system manager tells a colleague about her idea. The colleague, however, points out that the system manager's intended program will not work. The colleague argues that, while a few aspects of the idea are sound, it cannot be achieved as the system manager intends. Why would the colleague feel this way? I.e. why will the young system manager's program not work?

2. A student has composed the following program, and stored it in a file named `test.c`:

```
#include <stdio.h>
main()
{
    printf( "Hello World!\n" );
}
```

He compiles and tests the program on one of the *tonka* machines in the NetBSD lab as follows (">" is

the *csh* prompt for the user):

```
> gcc test.c
> a.out
Hello World!
>
```

However, he subsequently notices the following apparently erroneous behaviour:

```
> ls test
test
> gcc -o test test.c
> test
>
```

I.e. nothing is output; his test program appears not to have worked. The student suspects that there is an error with the C compiler, *gcc*. As a UNIX expert, however, you suspect that there is nothing wrong with *gcc*, and that the behaviour has a different cause. What is the cause of the apparently erroneous behavior then? Why is the user seeing the behaviour described above? What commands you would issue to validate your hypothesis?

3 A student has written a small program to obtain a print out the user groups she is in. The main part of the program is as follows:

```
gid_t *gidset;
k=getgroups(0,gidset);
getgroups(k,gidset);
printf("number of groups:  %d\n",k);
for (d=0;d<k;d++)
  {
  printf("%d\n",gidset[d]);
  }
```

Unfortunately, the program aborts with a "segmentation violation" error. I.e. there is an error in the program. What is the error? The *man* page for *getgroups(3)* reads, in part, as follows:

```
NAME
     getgroups - get group access list

LIBRARY
     Standard C Library (libc, -lc)
```

```
SYNOPSIS
     #include <sys/types.h>
     #include <unistd.h>

     int
     getgroups(int gidsetlen, gid_t *gidset);

DESCRIPTION
     getgroups() gets the current group access list of the current user pro-
     cess and stores it in the array gidset.  The parameter gidsetlen indi-
     cates the number of entries that may be placed in gidset.  getgroups()
     returns the actual number of groups returned in gidset.  No more than
     NGROUPS_MAX (defined in <unistd.h>) will ever be returned.  If
     gidsetlen is 0, getgroups() returns the number of groups without
     modifying the gidset array.

RETURN VALUES
     A successful call returns the number of groups in the group set.  A value
     of -1 indicates that an error occurred, and the error code is stored in
     the global variable errno.
```

H. (4 marks)

Many tape drives provide a block-device interface. Is it possible to support a BSD "fast" file system on such a (tape) device? Would it be practical? Why or why not? Consider the characteristics of such a device and the types of access necessary to practically support a UNIX file system?

I. (6 marks)

Consider the *fseekdemo* program below. The program is designed to create a data file with one record for each of five users. For some strange reason, the programmer who wrote it designed the program to write the records of the file backwards (record 4 first, record 0 last). However, this odd manner of writing out the records does not change the ultimate effect of the program.

```
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

struct record {
   int uid;
   char login[9];
}

char *logins[] = { "user1", "user2", "user3", "user4", "user5" };

main()
{
   int i, fd;
   struct record rec;

   if( (fd = open( "datafile", O_WRONLY|O_CREAT, 0644 )) < 0 ) {
      perror( "datafile" );
      exit( 1 );
   }

   for( i = 4; i >= 0; i-- ) {
      rec.uid = i;
      strcpy( rec.login, logins[i] );
      putrec( fd, i, &rec );
   }

   close( fd );
   exit( 0 );
}

putrec( fd, i, r )
int i, fd;
struct record *r;
{
   lseek( fd, (off_t)(i * sizeof(struct record)), SEEK_SET );
   write( fd, (char *)r, sizeof(struct record) );
}
```

Using the *fseekdemo* program above as a guide, write a C program in the space below which will read the same file and print each record out as it is read. However, the program is to read the records in the order 3, 0, 2, 1, 4.

Hint: the following definitions in the referenced "include" files may be of use:

```
#define NULL   0  /* null pointer constant */

#define SEEK_SET  0  /* set file offset to offset */
```

```
#define SEEK_CUR  1  /* set file offset to current + offset */
#define SEEK_END  2  /* set file offset to EOF plus offset */

#define O_RDONLY  0x0000   /* open for reading only */
#define O_WRONLY  0x0001   /* open for writing only */
#define O_RDWR    0x0002   /* open for reading and writing */
#define O_CREAT   0x0200   /* create if nonexistent */
#define O_TRUNC   0x0400   /* truncate to zero length */
```

J. (4 marks)

Is it possible for external fragmentation to occur in the UNIX file system? If so, describe how the problem occurs, and what steps were taken in the BSD FFS (Fast File System) to address the problem.

Hint: fragments and fragment blocks were introduced to address internal fragmentation.

K. (4 marks)

What are the differences between a symbolic link and an AF_UNIX socket? What are the similarities?

L. (3 marks)

What analogies can be drawn between the bootstrap software (bootstrap loader) for a computer and the starter in a car? I.e. what are the similarities between a bootstrap and an automobile (engine) starter?

Epilog

That's it! You're all done!

I sincerely hoped that you learned a lot in this course, and will find the material covered in the course valuable in your future endeavors. Have a good summer!

Extra Space

(The space below is for answering previous questions or for rough work.)