Tutorial Week 15

1.a) no changes are made externally. b) x gets incremented twice, becaussse it is referenced twice from within the procedure. c) x gets incremented once. Each local copy of x is incremented, and then the two local copies are restored. (Ada is apparently ambiguous as to whether (b) or (c) is done in that language.)

2.The first block can be done any number of times, as it always starts at zero. Later blocks probably could not if the server keeps track of the file pointer, but could if the client did.

3.A copy would be made, changed, and replaced. Locally, it would probably have to be done the same way, with at least some of the inode info copied (eg block pointers). Some could be ignored, such as owner, date, etc.

4.The two clocks must be logically synchronised or this breaks down.

5.Locally, Unix uses the user–id. This would be meaningless on a remote system because user–ids may not be the same. The name would have to be used. This would need validation on the server side. If the client was ‘‘trusted’’ then that might be enough, if untrusted,a password may be needed. In a stateless system, some verification would be needed for each transaction. This would have to be fast.

Tutorial Week 14

1. In a connection–oriented system, all clients would have to connect to the server, which would be impractical as they would not know that a message was coming. Alternatively, the *server* would have to connect to all clients, assuming that it knew them all. So this looks ot so good. On the other hand, this would ensure that all clients did get tbe message, or at least that the server would know about it. In connectionless broadcast, the server would not know if anybody missed a message, and would not have any security on who received it. On the other hand, it would not have to do anything special.
2. It would have to poll each port in turn, looking to see if anything comes on any one of them. ie this must be non–blocking reads/selects from the port.
3. In a connectionless system there is no guarantee that messages will arrive in the order sent. So to ensure that a debit is done before the credit, a connection–oriented version would be used.
4. Encode the two–byte *xy* as the 4–byte *xyyx*. That way either way of reading will get *xy*. If the first word is an ‘‘order’’ flag eg 1001 for big, 0110 for little, then the receiver will be able to determine the byte order of the incoming message and hence whether or not to swap it.

Tutorial Week 13

1.Best case is when two nodes are adjacent. Worst case is when they are both at the bottom connected only by the root. In this case there are 2*height nodes to traverse. The worst case path is 2*height, but the number of nodes in a balanced tree is proportional to 2 *to the power of* height. The measn that the length is logarithmic in the number of nodes, which is pretty good. A square mesh for example is only proportional to the square root. The downside is that the root acts as a bottleneck as any message from the left to the right of the tree passes through the root.

2.The message originates in Oz and is destined for the US. So it must pass through the gateway in Melbourne. Once in the US, it belongs in the commercial domain, so its location can be found from a name server keeping tracks on this domain. Once in the Microsoft network their own name server can locate bill_gates mail account.

3.Advantages: speed, reliability. Disadvantages: ensuring the two file systems contain the same info. Often writes would be allowed to one only, and the other ‘‘mirrors’’ this one.

4.Class A networks use one–bit to signify this, so 2^{31} addresses are available to these. Class B use 2 bits to signify this, so 2^{30} addresses are available. Class C use 3 bits to signal class, so 2^{29} addresses are available. Class D allows 2^{28} addresses. All told – a lot! Neverhtheless, it is running out. It is to do with the structuring of addresses. There are 2^{16} (65536) Class B addresses, and within a Class B network there are 64k possible machines. We have a Class B, but only 1000 (?) machines on campus. So we waste a lot. So does everyone–else. Moving to Class D would leave us with too few addresses. So the wastage is the reason.

5.Run one compile per machine, gives a speedup of a maximum of *n*. It may be less because: one compile may be huge and the others tiny, so it uses all the time anyway; the files may need to be sent out to the other machines, and for short compiles this may not be worth it; linking would have to be performed back on one machine.

6.From a PC lab, use ftp to transfer files to and from Unix.

from the filename (or other processes would not be able to find the lock!). Attempt to create the file in read–only mode. If it succeeds, you have the lock. If it fails, you don't. Remove the lock file on exit. You will have to trap interrupts so that even if the user types ctrl–C you can still remove the lock. Your program should never crash of its own accord :–)

Tutorial Week 12

1.A h/w swap interchanges two memory locations as an indivisible operation. So you load one location with 1, swap and see what you had in the other location. If 0, then the semaphore was free, else in use.

```
DOWN(s)
   x=1
   SWAP(x, s)
   if x == 1
      place this process on sleeping queue

UP(s)
   s=0
   remove a process from sleeping queue and
        let it run DOWN again.
```

2.

```
DOWN(s)
   if TEST–AND–SET(s, 1) == 1
      place this process on sleeping queue
UP(s)
    if there is a process on sleeping queue
      remove process from queue
    s=0
    let process run
```

3.Yes they can, but it means that a process at the bottom of the stack may suffer starvation.

4.When a DOWN is attempted, interrupts are switched off. The process can then examine a piece of memory to see what its value is, and reset it to another value if desired. Interrupts are then switched back on and the proces continues. Note that they should only be switched off while the semaphore is being examined/modified. If you just tried keeping them off all the time the process executes, then effectively only one semaphore is allowed.

5.Deadlock may occur if A's account is locked to debit it, and then B's account is locked. In the meantime, there may be a transfer the *other* way that results in the lock being placed on B first. This can be avoided by either (a) releasing all current locks if you can't get the next one, (b) ordering the locks, say by account number

6.One semaphore can be used to control access to the head of the queue, and another to the tail. Conflict may occur when the queue is empty, as both refer to the same location, or full. At these extremes both semaphores may need to be held at once, and you need to decide on an order eg head first.

Laboratory Week 12

7.A suitable lock file is one based on the file name, eg made from the file name appended with ".lock" This name has to be unique, but can be generated uniquely

original code. It should do its work then call the original interrupt code itself. Note that TSRs have much potential to “misbehave”. They should allocate all required data space on startup, not throw away original interrupt code, etc. In terms of process models, the process that called the interrupt is “suspended” while the TSR executes.

Laboratory Week 9

1.

```
if ((pid = fork()) == 0) {
  execlp(command);
} else {
  while (wait(&status) != pid)
    ;
  return WEXITSTATUS(status);
```

2.

```
if ((pid = fork()) == 0 {
  while (1)
    printf(”Running forever\n”);
} else {
  sleep(5);
  kill(pid, SIGSTOP);
  sleep(5);
  kill(pid, SIGCONT);
  sleep(5);
  kill(pid, SIGINT);
}
```

Tutorial Week 9

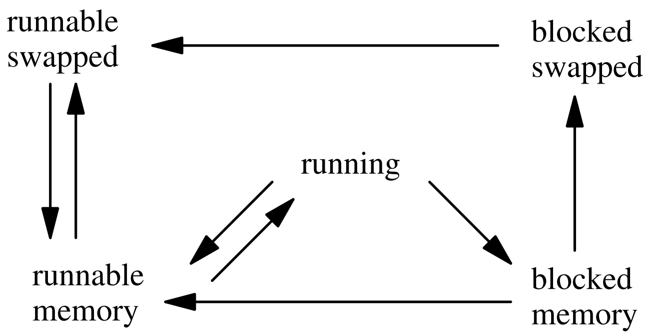
1. time action

time	action
0	P0 starts
20	P0 is stopped, with 50 remaining
21	P1 starts
41	P1 is stopped with 5 remaining
42	P2 starts
62	P2 is stopped with 25 remaining
63	P0 restarts
83	P0 is stopped with 30 remaining
84	P1 restarts
89	P1 terminates
90	P2 restarts
110	P2 is stopped with 5 remaining
111	P0 restarts
131	P0 is stopped with 10 remaining
132	P2 restarts
137	P2 terminates
138	P0 restarts
148	P0 terminates

There are 8 context switches in 148 units. So the switching overhead is 8/148 * 100% ~ 5%. The average to completion is (89 + 137 + 148)/3.

2.The sequence of running (with priorities is)
T3(18), T3(17), T3(16), T3(15), T3(14), T1(15), T1(14), T1(13), T3(13), T3(12), T1(12), T1(11), T3(11), T3(10), T1(10), T1(9), T2(9), T3(9), ... as in round robin

3.



4.Round robin: starvation cannot occur because a process will eventually reach the fornmt of the queue; shortest–first: a long process couyld be starved if shorter ones can be continually added; priority: yes, if higher priority jobs keep getting added that run for short enough that their priority never falls low enough.

5.A refernence count should be used for each piece of memory. When a new process starts using a piece of memory the count is increased, and decreased on termination. When the count is zero, it can be removed.

6.A TSR is loaded in the normal way. Upon exit, its memory space is not recalimed but is left there (stay resident). When first loaded a TSR replaces the code for some interrupt so taht the TSR is trigered when the interrupt occurs instead of the

Tutorial Week 8

1.Internal fragmentation is when a process is given memory and is not using all of it, as in the buddy system. External fragmentation is when their are lots of small entries on the free list that cannot easily be used.

2.First fit: 20k, 10k, 18k, 8k hole left in 20k hole; left– 4k, 9k, 7k, 9k, 12k, 15k
Best fit: 12k, 10k, 9k, 15k; left – 4k, 20k, 18k, 7k, 7k
Worst fit: 20k, 18k, 15k; left – 10k, 4k, 8k, 8k, 7k, 9k, 8k, 6k

3.All of them. Because of the horrible memory model. You may need a different strategy for badly fragmented upper memory to ordinary memory.

4.

```
if new_size == old_size
    return old_ptr
if new_size < old_size
    free spare blocks at the top
    return old_ptr
if there is a free block size (new_size – old_size)
    above and contiguous to the old one,
    remove this from free list
    return old_ptr
get a block from free list of new_size
set new_ptr to it
copy old to new
return new_ptr
```

5.LA 20 → PA 02020; LA 4100 → PA 05100; LA 010300 → PA 04300; LA 05376 → page frame fault.

```
#include <dirent.h>

int main(void)
{
  DIR *dp;
  struct dirent *direntry;

  if ((dp = opendir(".")) == NULL) {
    fprintf(stderr, "cant open .\n");
    exit(1);
  }

  while ((direntry = readdir(dp)) != NULL)
    puts(direntry->d_name);

  closedir(dp);
  exit(0);
}
```

7.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/param.h>

int main(void)
{
  struct stat buf;
  char name[MAXPATHLEN];

  printf("Enter a file name:");
  gets(name);

  if (stat(name, &buf) != 0) {
    fprintf(stderr, "cant stat %s\n", name);
    exit(1);
  }

  if (S_ISDIR(buf.st_mode))
    printf("%s is a directory\n", name);
  else
    printf("%s is not a directory\n", name);

  exit(0);
}
```

Tutorial Week 7

1.a) the user sees a hierarchical file system with mountable file systems (ie they can stick floppies in and out). File names have a maximum length that is too short, and there are no real security measures to protect them against carelessness, viruses (viri?) etc.
b) What they need to undestand is the example program at the end of the _dos_find function descriptions, because that is how they traverse directories in MSDOS. It is different to Unix, although it accomplishes much the same thing. _dos_findfirst both does an open of the directory and also does a read to find the first file. Note that this also allows pattern matching of file names. The description also mentions an implementation detail: this uses an interrupt to MSDOS to do the work. The information returned corresponds to the MSDOS directory information and gives you all that the directory has. (In contrast, in Unix you have to make two calls to get file info because such info is stored in inodes, not in the directory.)
c) Check that they can find out how to read say the second block of a file, given a pathname such as C:\windows\win.com

2.When a file is deleted, the first character of the file name is overwritten to show that the file is deleted. The actual blocks are not cleared. This allows the utility to restore the file as long as none of the freed blocks have been overwritten by later calls. This works because MSDOS users often ‘‘freeze’’ when they have incorrectly removed a file so it has a good chance of not being overwritten. In a timesharing environment this will not help, because the longer you wait the more chance there is that another process overwrites the file. Undelete could only work by making a copy in a secret place and then deleting it later. Undelete would just restore from the copy.

3.It would slow performance because all blocks would have to be cleared. It is a security hole if they are not cleared because the information may still remain. It depends on your secrecy requirements (eg drug dealers would probably want full erasure, the cops don’t).

4.Copying takes extra time. It also uses up extra file space while the copy is made. This may make it impossible to rename a file this way.

5.Fixed length makes implementation easy because you can use fixed size directory records. This makes deleting and reusing the space easy. It also allows easier searching, because you can get from one record to the next quickly. It would be *very* tedious for a user if every filename had to have the same number of characters in it! A maximum length would have the same advantages for implementation if stored as fixed size and would be nicer for the user, but would waste disk space. If variable sized records were used (they would have to be in the first case) then deletion of files would leave variable–sized ‘‘holes’’ in the directory that would require compaction at times.

Laboratory Week 7

6.This is for the current directory:

```
#include <stdio.h>
```

```
  }
  exit(0);
}
```

3.Each command is handled by a new shell – chdir cahnges for the new shell which then exits, so it has no effect on the current environment. You can combine the last three solutions to get a simple interpreter of your own.

```
#include <stdio.h>
int main(void)
{  char path[2048]; /* they'll never type this much... */

  while (gets(path) != NULL)
    system(path);
  exit(0);
}
```

```
int main(void)
{
   char *strings[MAX];
   char str[1024];          /* hope this is long enough */
   int count = 0;
   int n;

   while (count < MAX && scanf("%1023s", str) != EOF) {
      strings[count] = malloc(strlen(str) + 1); /* space for null char */
      strcpy(strings[count], str);
      count++;
   }
   for (n = 0; n < count; n++)
      puts(strings[n]);
   exit(0);
}
```

Laboratory Week 6

1.

```
#include <sys/params.h>
#include <stdio.h>
int main(void)
{
   char dir[MAXPATHLEN];

   if (getwd(dir) == NULL) {
      fprintf(stderr, "cant find current dir\n");
      exit(1);
   }
   puts(dir);
   exit(0);
}
```

2.This will recognise ., .. patterns but not ~ – only the shells do that one too. MAXPATHLEN is used again to give a reasonable size.

```
#include <stdio.h>
#include <sys/params.h>

int main(void)
{
  char path[MAXPATHLEN];

  while (gets(path) != NULL) {
   if (chdir(path) == 0)
     printf("changed to %s\n", path);
   else printf("failed to change to %s\n");
```

Tutorial Week 6

1.

```
#include <stdio.h>
#define MAXNUM 100

int main(int argc, char *argv[])
{
   int numbers[MAXNUM];
   int number, count;

   count = 0;
   while (count < MAXNUM && scanf(”%d”, &number) != EOF) {
      numbers[count++] = number;
   }
   while (count > 0)
      printf(”%d\n”, numbers[—count];
   exit 0);
}
```

2.

```
int strcmp(char *s1, char *s2)
{
   while (*s1 != ’\0’ && *s2 != ’\0’) {
      if (*s1 < *s2) return –1;
      if (*s1 > *s2) return 1;
      s1++;
      s2++;
   }
   if (*s1 != ’\0’) return 1;    /* s1 still has chars */
   if (*s2 != ’\0’) return –1;   /* s2 still has chars */
   return 0;
}
```

3.

```
#include <stdio.h>

struct student_t {
   char *name[100];
   int ID;        /* will need to be long if int is 16 bits */
} students[MAXSTUDENT];
```

etc

4. If you don’t use malloc, but just use one string, everything in the array will end up pointing to the string.

```
#include <stdio.h>
#define MAX 100
```

```
{
   if ('A' <= ch && ch <= 'Z')
      return (ch + 'a' – 'A');
   return ch;
}
```

8.The type should probably be long for a real factorial function.

```
int factorial(int n)
{
   if (n == 0)
      return 1;
   return n * factorial(n – 1);
}
```

Laboratory Week 5

1.Look in the files /usr/include/limits.h. What are the largest and smallest integers that are supported? Write and test a program to print out their values.

2.It crashes because the pointer is uninitialised and is probably pointing to junk

Tutorial Week 5

1.At the low level: ability to read/write to absolute hardware addresses; use data types that correspond to machine hardware types (unsigned/signed ints, etc); be able to link with Assembler routines; ability to have inline Assembler.
At the high level: be able to offer a high–level API to applications.

2.It must be able to link with the O/S libraries.

3.Of course not – MSDOS is written in Assembler, but most programs are in Pascal or C. You may lose the ability to exploit all of the O/S services if, for example, the application programming language is unable to use the data types in the O/S API.

4.The obvious differences are syntactic, with operators like += in C, {...} instead of LOOP...END LOOP, etc. The Ada exmple shows size formatting in the output – the C printf can also do that, but it isn't shown. Ada does not have a predefined entry point to a program so it defines a Procedure, whereas C has to have ‘‘main’’ as entry point. Ada uses the IO packages, and the Ada environment should elaborate these automatically on linking. In C, the standard library is automatically linked, and an O/S specific linker would also link in the O/S kernel libraries. However, any other libraries may have to be linked explicitly. Note: I am using the ‘‘Indian Hill’’ style for programs, and so is the recommended reading of Kelley and Pohl. Use whatever style you want, but be consistent.

5.I use the library functions here cos I am lazy. The studnets may be better off expanding them out. *NB* Use ANSI C, not K&R C. This is a change from last year.

```
#include <stdio.h>
int main(int argc, char *argv[])
{  int ch;
   while ((ch = getchar()) != EOF) {
      if (isupper(ch)) putchar(tolower(ch));
      else if (islower(ch)) putchar(toupper(ch));
         else putchar(ch);
   }
   exit(0);
}
```

6.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int ch;

   while ((ch = getchar()) != '\n')
      if (! isdigit(ch)) exit(1);
   exit(0);
}
```

7.

```
int tolower(int ch)
```

Tutorial Week 4

1.command.com has ‘‘batch’’ processing commands ECHO, FOR, GOTO, IF, PAUSE, REM, SHIFT. This is smaller than the zsh set. command.com does not have ‘grave’ redirection. If a .bat file is executed from within another, it does not return to the first one on completion. command.com can use ERRORLEVEL, string equality, EXIST and NOT in IF commands. Apart from these, they have quite similar functionality.
command.com handles pipelines by running each command sequentially from left to right, storing output in temporary files along the way.
There already exist Ada equivalents, which are Ada interpreters with additional bits. The strong typing of Ada means that you have to declare all types and variables as you go, making it a bit more cumbersome.

2.Keyword advantages: easier to understand, don’t have to remember meanings of positions. Disadvantages: more cumbersome to type.

3.
```
echo $*
read answer
case $answer in
   Y*)) exit 0;;
   y*)) exit 0;;
   *)) exit 1;;
esac
```

4.
```
if ask ”remove file $i”
then
   rm $i
fi
```

5.
```
file=$1
shift
for i
do
   grep $i $file
done
```

Laboratory Week 4

1.

2..profile – read by zsh at login time, is the startup shell script
.zshrc – a shell script read and executed by any zsh when it starts.
.xinitrc – a shell script read and executed by X when it starts
.mwmwrc – the control file for the mwm window manager
.Xdefaults – the general control file for all X applications.

Tutorial Week 3

1.Shell script file contains

```
ls –l
```

2.ls | wc –l

3.cat * | wc –c

4.grep $1 $2 | head –1

5.ls $1 > /tmp/tmp1$$

```
ls $2 > /tmp/tmp2$$
diff /tmp/tmp1$$ /tmp/tmp2$$
rm –f /tmp/tmp*
```

6.find . –name '*' –exec grep –l $1 {} \;

7.new=‘echo $1 | tr "A–Z" "a–z"‘

```
mv $1 $new
```

Operating Systems
and Computer Structures G2

Tutorial Week 2

1. To provide management of hardware resources, and to provide management of software resources (programs, files, etc).
2. To allow orderly processing of jobs, so that they can be printed in some sort of order and without trying to simultaneously access the hardware.
3. Multi–user; multi–tasking; security; address capability; batch processing
4. Shell functionality is not inherent in the O/S. It allows different shells to be used.
5. Video cards of different types; floppy disks of different sizes. Need a common abstraction above this (open, write, close) to hide the differences.
6. DIR, COPY, REN, TYPE, DEL, CD, MKDIR, RMDIR. Options are signalled by /option instead of –option. Range of options is usually smaller.
7. MS Windows: No; Windows NT: Yes. NT: flat memory model, support for multi–tasking.
8. Use the File Manager.
9. Unix performs globbing in the shell, MSDOS by individual applications. Unix allows as many wildcards as wanted in the pattern, MSDOS does not.

Laboratory Week 2

1.

2.

3.

4. at – run a command at a later time
   compress – compress a file to use less space
   du – report on disk usage
   file – find the type of a file (program, C text, etc)
5. –R – recursive listing
   –t – order by time modified instead of alphabetic
   –rta – all files (including starting with .), reversed time order.

For: jan
Printed on: Wed, Aug 23, 1995 11:45:15

Document: tuts-sols-all
Last saved on: Thu, Jul 14, 1994 17:05:15