CSCI 493.66 Unix Tools

with emphasis on string algorithms and Bioinformatics

Homework 2
Due 02/26/09

Saad Mneimneh
Computer Science
Hunter College of CUNY

Problem 1: who
PART A: for C/C++ programmers.

In class I presented a C++ implementation of a basic who program using the C++ IO stream library:

```
#include <iostream>
#include <fstream>

using namespace std;

#include <utmp.h>

void show_info(struct utmp * current_record) {
  cout<<current_record->ut_name<<" "<<current_record->ut_line<<" "
      <<current_record->ut_time<<" "<<current_record->ut_host<<"\n";
}

int main() {
  struct utmp current_record;
  ifstream utmpfile(UTMP_FILE, ios::in | ios::binary);

  while(true) {
    utmpfile.read((char *)&current_record, sizeof(current_record));
    show_info(&current_record);
    if (utmpfile.eof()) {
      break;
    }
  }
  utmpfile.close();
}
```

The book implements the same program using C and the system calls open, read, and close that deal with the notion of *file descriptors*.

(a) Implement the who program above using the C standard IO library. Make sure to display only entries corresponding to a user process and to use the ctime() function to properly convert the time to a human readable string. As a

side exersice, find out why it is important to use ctime() only upon printing the time value, in other words, it is not a good idea to obtain the string, and then print it later on.

(b) Do some research online on the differences between the standard IO library, e.g. fread(), and the low level system calls that use file descriptors, e.g. read(). Compare both implementations for speed. You can use the function time() to obtain the time before and after IO access, but you might want to repeatedly access the utmp file in a loop to notice a difference.

(c) Modify who in such a way to read all entries in the utmp file and store them in an array. For this, you want to identify the size of the array in advance. Use fseek() and ftell() to determine that size (you will also need to know the size of a utmp struct). Once all the entries are in the array, sort them based on the time field for each entry. Then display the information in the sorted order. You can use qsort() in stdlib.h (or cstdlib if using C++ for this part). Look online for the specification of this function and how it should be used. You will have to implement on the side a comparator function that compares two utmp entries.

PART B: for Perl programmers.

(a) Use Perl to read C strcut entries from the utmp file as described in class. For this you have to figure out the size of each field in bytes, and also the extra bytes used for byte alignment (these will be ignored of course). Here's the C struct for utmp:

```
struct utmp {
  short int ut_type;
  pid_t ut_pid;
  char ut_line[32];
  char ut_id[4];
  char ut_user[32];
  char ut_host[256];
  struct exit_status ut_exit;
  long int ut_session;
  struct timeval ut_tv;
  int32_t ut_addr_v6[4];
  char __unused[20];
};
```

where the following two structs are also defined:

```
struct timeval {
  __time_t tv_sec;            /* Seconds.  */
  __suseconds_t tv_usec;      /* Microseconds.  */
};

struct exit_status {
  short int e_termination;
  short int e_exit;
};
```

You need to use the unpack() function to retrieve the entries based on their format. Look for the unpack() function in Perl.

(b) You are asked to maintain an array of hashes. Each entry read should be stored in a hash structure where the keys are given by the names of the fields in the C struct (structures inside the utmp struct should be expanded).

(c) Display the information in the array. Only entries that corresponds to a user process should be displayed, i.e. ut_type is equal to 7. You can also use scalar(localtime(tv_sec)) to obtain a string corresponding to the time value.

Problem 2: ls

Here's a C++ code for a basic ls program:

```
#include <iostream>
#include <dirent.h>
#include <sys/stat.h>

using std::cout;

void do_ls(char * dirname) {
  DIR * dirp;
  dirent * direntp;
  if ((dirp=opendir(dirname))==NULL)
    cout<<"cannot open "<<dirname<<"\n";
  else {
    while ((direntp=readdir(dirp))!=NULL) {
      cout<<direntp->d_name<<" ";
      struct stat info;
      stat(direntp->d_name, &info);
      cout<<info.st_mode<<" ";
      cout<<info.st_nlink<<" ";
      cout<<info.st_uid<<" ";
      cout<<info.st_gid<<" ";
      cout<<info.st_size<<" ";
      cout<<info.st_mtime<<"\n";
    }
    closedir(dirp);
  }
}

int main(int ac, char ** av) {
  if (ac==1)
    do_ls(".");
  else
    while (--ac) {
      cout<<*++av<<"\n";
      do_ls(*av);
    }
}
```

(a) Modify this program to display the above information in a human readable

format, and also to recursively list the entries in each subdirectory.

(b) Modify this program to display the above information in a human readable format, and also to group files of the same type together. For this, you can insert entries in a priority queue based on their file type (obtained from the d_name field), and then repeatedly retrieve entries from the queue until the queue is empty. We will discuss this in class. There are also other possible alternatives.