Primer Certamen: Tiempo: 11:45 hrs - 13:30 hrs.
Todas las preguntas tienen igual puntaje.

1.- Windows no diferencia entre mayúsculas y minúsculas en nombres de archivos. Así al crear páginas web es común que éstas funcionen bien en servidores windows pero no en servidores corriendo sistemas operativos derivados de Unix.
Desarrolle toLower.sh, un script shell que pasa a minúscula todos los nombres de archivos bajo un directorio dado.
La ejecución del script es:
$toLower.sh <Directorio>
Por ejemplo, lo podemos correr como en $ toLower.sh .
Ayuda: Para pasar un string a mayúscula en shell usted puede declararlo usando:
declare -u nombre_de_su_variable="Aquí lo que usted pasará a mayúscula"
declare -l nombre_de_su_variable="Aquí lo que usted pasará a minúscula"

```
#! /bin/bash
case $# in
0) dir=. ;;
1) dir=$1 ;;
esac

echo .... scanning directory $dir
declare -l lowerCase=$dir
if test "$dir" != "$lowerCase"
  then
    mv $dir $lowerCase
fi
for i in $lowerCase/*
do
        if test -d $i
        then
                $0 $i
        else
                declare -l lowerCase=$i
                if test "$i" != "$lowerCase"
                  then
            mv $i $lowerCase
         fi
        fi
done
```

2.- Haga un programa en C que obtenga el número máximo de procesos simultáneos adicionales que el sistema admitirá en ese momento. Antes de terminar su programa muestra tal número. Posibles procesos creados en esta búsqueda deben terminar.

Cualquier solución que intente buscar este límite experimentalmente es OK como respuesta.

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

```
#include <unistd.h> /* for write*/
#include <stdlib.h> /* for exit */
#include <sys/time.h>
#include <sys/wait.h>

struct timeval t0;

/* count: number of processes so far */
/* return value: number of total processes */
void newProcess(int count) {
   int pid;
   struct timeval time;

   count++;
   if (count%10==0){  /* just to monitor progress */
      gettimeofday(&time, NULL);
      printf("At time [us], %ld,%i, processes have been created\n",
            (time.tv_sec-t0.tv_sec)*1000000+
            (time.tv_usec-t0.tv_usec),count); fflush(stdout);
   }
   if ((pid = fork()) < 0){
      printf("I was able to create %i processes\n", count);
      return;
   } else if (pid == 0){  /* child*/
      newProcess(count);
      exit(0);
   }
    /* father */
   waitpid(pid,NULL,0);
}

int main(void)
{
   int count=0;
   gettimeofday(&t0,NULL);
   newProcess(count);
}
```

Notar que algunas soluciones pueden conducir el sistema a "thrashing" (concepto de sistemas operativos) antes de generar error en el llamado a fork(). En mi notebook, la solución mostrada pudo crear 15.536 procesos y demoró poco más de 9 minutos.

3.- Cree myTCPeco, un servidor TCP que retorne al cliente todos los datos recibidos desde él, siempre y cuando el cliente provenga de una máquina en un dominio específico señalado en la línea de comandos. Se trata de un servidor de eco que sólo atiende al primer cliente y luego termina su ejecución. En caso de tratarse de un cliente desde un dominio distinto al permitido, el servidor cierra la conexión y termina.
Ejemplo, al ejecutar:
$ myTCPeco elo.utfsm.cl

Se les enviará el eco sólo a los clientes que proviene del dominio elo.utfsm.cl.
Nota: No se pide entregar el código del cliente.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <netdb.h>

int main(int argc, char * argv[])
{
   char buf[256];
   int s, n, ns, len;
   struct sockaddr_in server, client;
   char * validDomain = argv[1];
   char * clientDomain;
   struct  hostent *hp;

   s = socket(AF_INET, SOCK_STREAM, 0);
   server.sin_family = AF_INET;
   server.sin_port = htons(0);
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   len = sizeof(struct sockaddr_in);

   bind(s, (struct sockaddr *) &server, len);

   /* find out the port number given by the OS */
   getsockname (s, (struct sockaddr *) &server,&len);
   printf("Server Port is: %d\n", ntohs(server.sin_port));
   listen(s, 5);
   ns = accept(s, (struct sockaddr *) &client, &len);
   /* find out the client hostname */
   hp = gethostbyaddr((char*)&client.sin_addr.s_addr,
                sizeof(client.sin_addr.s_addr),AF_INET);
   clientDomain=hp->h_name;
   while (((*clientDomain) != '.') && ((*clientDomain) != '\0'))
     clientDomain++;
   if ((*clientDomain) == '.') clientDomain++;  /* skip the '.' */
   if (strcmp(validDomain,clientDomain)==0)
     /* Read from the socket until connection is close and
       send back what we get.  */
     while ((n = read(ns, buf, sizeof(buf))) > 0)
       write(ns, buf, n);
   close(ns);
   close(s);
}
```