Grupo CSI-81		1151018	Sistemas Operativos
Documento	Proyecto final		
Fecha	9-IV-18		
Trimestre	T-18I		
Actividad: **SIMULADOR DE UNA HERRAMIENTA DE ADMINISTRACIÓN DE UNIX**			
Calificación S			

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <ifaddrs.h>
#include <netdb.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <time.h>

enum tokens
{
   EOL = 1,
   ARG,
   AMPERSAND,
   PUNTOYCOMA,
   PIPE
};

#define MAXARG 512
#define MAXBUF 512

enum plano
{
   PRIMERPLANO,
   SEGUNDOPLANO
};

typedef enum { FALSO,
            CIERTO } bool;
```

```
bool esargumento(char);
int sistema(char **, int);
void procesalinea(char *);
int dametoken(char **, char *, char *, char **, char **);
int leerlinea(char *, char *);
void registraCMDS(char *comando, int pid);
void registraError(char *comando, int pid);
void imprimeB(char *archivo);
void configurar(int i);
void error(char *);
void procesapipe(char *, char **);
int juntar(char *comando1[], char *comando2[]);/*
Autores:
Alumno 01                   Matricula 01
Alumno 02          Matricula 02
Alumno 03                   Matricula 03
Alumno 04                      Matricula 04
Alumno 05          Matricula 05
*/

/*
Encargado de desplegar opciones de configuracion
y guardar los cambios en el archivo de configuacion
*/

#include "uamashell.h"

void configurar(int numline)
{
	FILE *file;
	char newvalue;
	char* newline;
	char lineaux[250];
	printf("Registre el nuevo valor:");
```

```
if(numline==1)
{
newvalue=getchar();
newvalue=getchar();
}
else
{
scanf("%s",newline);
}
int i=0;
while(newline++!=NULL)
{
lineaux[i]=newline[i];
i++;
}
char buffer[3][255];

i=0;
int j=0;
file=fopen("../conf/config","r");
char ch=getc(file);
while(ch!='@')
     {

       buffer[j][i]=ch;
       i++;
       if(ch=='\n')
       {
        buffer[j][i]='\0';
         j++;
              i=0;
       }
       ch=getc(file);
     }
fclose(file);
```

```
	file=fopen("../conf/config","w");
	if(numline==1)
	{
		buffer[0][11]=newvalue;
		buffer[0][12]='\n';
		fputs(buffer[0],file);
		fputs(buffer[1],file);
		fputs(buffer[2],file);
		fputs("@",file);
	}
	if(numline==2)
	{

		fputs(buffer[0],file);
		fputs(lineaux,file);
		fputs(buffer[2],file);
		fputs("@",file);
	}
	if(numline==3)
	{
		fputs(buffer[0],file);
		fputs(buffer[1],file);
		fputs(lineaux,file);
		fputs("@",file);
	}
	fclose(file);
	printf("Registro guardado con exito\n");
}


/*
Autores:
Alumno 01		Matricula 01
Alumno 02	Matricula 02
Alumno 03		Matricula 03
```

```
Alumno 04                    Matricula 04
Alumno 05          Matricula 05
*/

/*
parsea y tokeniza la linea de entrada
*/

#include "uamashell.h"

int dametoken(char **apuntadorsalida, char *bufferentrada, char *buffertoken, char **apuntador, char **token)
{
  int tipo;
  *apuntadorsalida = *token;

  /* Saltarse los blancos */
  for (; **apuntador == ' ' || **apuntador == '\t'; (*apuntador)++)
    ;
  *(*token)++ = **apuntador;

  switch (*(*apuntador)++)
  {
  case '\n':
    tipo = EOL;
    break;
  case '&':
    tipo = AMPERSAND;
    break;
  case ';':
    tipo = PUNTOYCOMA;
    break;
  case '|':
    tipo = PIPE;
    break;
```

```
 default:
   tipo = ARG;
   while (esargumento(**apuntador))
     *(*token)++ = *(*apuntador)++;
   break;
 }

 *(*token)++ = '\0';
 return (tipo);
}/*
Autores:
Alumno 01                   Matricula 01
Alumno 02          Matricula 02
Alumno 03                   Matricula 03
Alumno 04                      Matricula 04
Alumno 05          Matricula 05
*/

/*
imprime mensaje de error
*/

#include "uamashell.h"

void error(char *mensaje)
{
       fprintf(stderr, "\a");
       strcat(mensaje, " Command Failure\n");
       //printf("UAM>");
       exit((errno) ? errno : -1);
}
/*
Autores:
Alumno 01                   Matricula 01
Alumno 02          Matricula 02
```

```
Alumno 03                   Matricula 03
Alumno 04                   Matricula 04
Alumno 05          Matricula 05
*/

/*
verifica sis el caracter leido
es un metacaracter del shell
*/

#include "uamashell.h"

bool esargumento(char c)
{
      char *trabajo;
      char especial[] = {' ', '\t', '&', ';', '\n', '\0', '|'};

      for (trabajo = especial; *trabajo; *trabajo++)
            if (c == *trabajo)
            {
                  return (FALSO);
            }

      return (CIERTO);
}
/*
Autores:
Alumno 01                   Matricula 01
Alumno 02          Matricula 02
Alumno 03                   Matricula 03
Alumno 04                   Matricula 04
Alumno 05          Matricula 05
*/

/*
```

```
imprime progrma.log
imprime programa_error.log
*/

#include "uamashell.h"

void imprimeB(char *archivo)
{
	FILE *bitacoraC;
	bitacoraC = fopen(archivo, "r");
	char line[512];
	int contador = 0;
	while (fgets(line, sizeof(line), bitacoraC))
	{
		printf("%s", line);
		contador++;
		if (contador == 22)
		{
			printf("Press Enter to continue...");
			while (getchar() != '\n')
				;
			contador = 0;
		}
	}
	fclose(bitacoraC);

	return;
}
/*
Autores:
Alumno 01		Matricula 01
Alumno 02		Matricula 02
Alumno 03		Matricula 03
Alumno 04		Matricula 04
Alumno 05		Matricula 05
```

```
*/

/*
crea el pipe entre dos comandos
*/

#include "uamashell.h"

int juntar(char *comando1[], char *comando2[])
{
   int p[2], estado;
   /* Crear hijo para correr comandos */
   switch (fork())
   {
   case -1:
      error("Fallo el 1er fork");
      break;
   case 0:
      break;
   default:
      wait(&estado);
      return (estado);
   }
   if (pipe(p) < 0)
      error("Fallo el pipe");
   /* Crear el nieto */
   switch (fork())
   {
   case -1:
      error("fallo el 2do fork");
      break;
   case 0:
      /* Cerrar salida estandar */
      close(1);
      /* Conectar la salida estandar al pipe de salida */
```

```
        dup(p[1]);
        /* Cerrar descriptores de archivo que no se usaran mas */
        close(p[0]);
        close(p[1]);
        /* Ejecutar el comando 1 */
        execvp(comando1[0], comando1);
        error("Fallo el 1er execvp ");
        break;
    default:
        /* Cerrar entrada estandar */
        close(0);
        /* Conectar la entrada estandar al pipe de entrada */
        dup(p[0]);
        /* Cerrar descriptores de archivo que no se usaran mas */
        close(p[0]);
        close(p[1]);
        /* Ejecutar el comando 2 */
        execvp(comando2[0], comando2);
        error("Fallo el 2do execvp ");
        break;
    }
}/*
Autores:
Alumno 01                   Matricula 01
Alumno 02          Matricula 02
Alumno 03                   Matricula 03
Alumno 04                      Matricula 04
Alumno 05          Matricula 05
*/

/*
lee la linea completa escrita
*/

#include "uamashell.h"
```

```
int leerlinea(char *prompt, char *bufferentrada)
{
	int caracter, contador;
	/* Mostrar prompt*/
	printf("%s ", prompt);
	for (contador = 0;;)
	{
		if ((caracter = getchar()) == EOF)
			return (EOF);

		if (contador < MAXBUF)
			bufferentrada[contador++] = caracter;

		if (caracter == '\n' && contador < MAXBUF)
		{
			bufferentrada[contador] = '\0';
			return (contador);
		}

		/* Si la linea es muy larga resetear */
		if (caracter == '\n')
		{
			printf("UamaShell: linea muy larga\n");
			contador = 0;
			printf("%s ", prompt);
		}
	}
}
/*
Autores:
Alumno 01		Matricula 01
Alumno 02		Matricula 02
Alumno 03		Matricula 03
Alumno 04		Matricula 04
```

```
Alumno 05          Matricula 05
*/

/*
avanza el apuntador de la linea leida
determinar el token
y la accion a ejecutar
*/

#include "uamashell.h"

void procesalinea(char *bufferentrada)
{
   char buffertoken[2 * MAXBUF];
   char *apuntador = bufferentrada;
   char *token = buffertoken;
   int tipotoken;
   int contador;
   int tipo;

   int i;

   char **arg = (char **)malloc(MAXBUF * sizeof(char **));
   if (arg == NULL)
   {
      perror("fallo malloc\n");
      exit(EXIT_FAILURE);
   }
   for (i = 0; i < MAXARG; i++)
   {
      arg[i] = (char *)malloc(MAXARG * sizeof(char));
   }

   char **comandoUno = (char **)malloc(MAXBUF * sizeof(char **));
   if (arg == NULL)
```

```
    {
        perror("fallo malloc\n");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < MAXARG; i++)
    {
        comandoUno[i] = (char *)malloc(MAXARG * sizeof(char));
    }

    char **comandoDos = (char **)malloc(MAXBUF * sizeof(char **));
    if (arg == NULL)
    {
        perror("fallo malloc\n");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < MAXARG; i++)
    {
        comandoDos[i] = (char *)malloc(MAXARG * sizeof(char));
    }

    for (contador = 0;;)
    {
        /* Actuar conforme al tipo de tokenen */
        switch (tipotoken = dametoken(&arg[contador], bufferentrada, buffertoken, &apuntador,
&token))
        {
        case ARG:
            if (contador < MAXARG)
                contador++;
            break;
        case EOL:
        case PUNTOYCOMA:
        case AMPERSAND:
            tipo = (tipotoken == AMPERSAND) ? SEGUNDOPLANO : PRIMERPLANO;
            if (contador)
```

```
            {
               //arg tiene el comando que se escribio en linea de comando
               arg[contador] = NULL;
               sistema(arg, tipo);
            }
            if (tipotoken == EOL)
               return;
            contador = 0;
            break;
         case PIPE:
            //guardar el primer comando ya leido
            //correr procesa linea apuntando despues del caracter pipe
            for (i = 0; i < contador; i++)
            {
               strcpy(comandoUno[i], arg[i]);
            }
            //comandoUno[contador] = NULL;
            procesapipe(apuntador, comandoDos);
            juntar(comandoUno, comandoDos);
            return;
         default:
            break;
            //break;
         }
      }
}
/*
Autores:
Alumno 01                  Matricula 01
Alumno 02          Matricula 02
Alumno 03                  Matricula 03
Alumno 04                     Matricula 04
Alumno 05          Matricula 05
*/
/*
```

guarda el segundo commando
que sigue despues del pipe
*/

#include "uamashell.h"

void procesapipe(char *bufferentrada, char **comandoOutput)
{
   char *arg[MAXARG + 1];
   char buffertoken[2 * MAXBUF];
   char *apuntador = bufferentrada;
   char *token = buffertoken;
   int tipotoken;
   int contador;
   int tipo;

   for (contador = 0;;)
   {
      /* Actuar conforme al tipo de tokenen */
      switch (tipotoken = dametoken(&arg[contador], bufferentrada, buffertoken, &apuntador, &token))
      {
      case ARG:
         if (contador < MAXARG)
            contador++;
         break;
      case EOL:
      case PUNTOYCOMA:
      case AMPERSAND:

         //tipo = (tipotoken == AMPERSAND) ? SEGUNDOPLANO : PRIMERPLANO;
         if (contador)
         {
            //arg tiene el comando que se escribio en linea de comando

```
            int i;
            for (i = 0; i < contador; i++)
            {
               strcpy(comandoOutput[i], arg[i]);
            }
            //arg[contador] = NULL;
            //comandoOutput[contador] = NULL;
            //sistema_tipo(arg, tipo);
         }
         if (tipotoken == EOL)
            return;
         break;
      default:
         break;
      }
   }
}
/*
Autores:
Alumno 01                  Matricula 01
Alumno 02          Matricula 02
Alumno 03                  Matricula 03
Alumno 04                     Matricula 04
Alumno 05          Matricula 05
*/

/*
registra los comandos ejecutados
en el archivo programa.log
*/

#include "uamashell.h"

void registraCMDS(char *comando, int pid)
{
```

```
    //Variables para abrir archivo bitacora_comandos.log
    FILE *bitacoraC;
    bitacoraC = fopen("../conf/programa.log", "at");
    //Obtener usuario
    char *user;
    user = getlogin();
    //Obtener datetime
    char date[20];
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    sprintf(date, "%d-%d-%d %d:%d:%d", tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);
    //PID
    char idp[16];
    sprintf(idp, "Process ID:%d", pid);
    //Obtener IP
    char *defaultIP = "127.0.0.1";
    FILE *f;
    char line[100], *p, *c;

    f = fopen("/proc/net/route", "r");

    while (fgets(line, 100, f))
    {
        p = strtok(line, " \t");
        c = strtok(NULL, " \t");

        if (p != NULL && c != NULL)
        {
            if (strcmp(c, "00000000") == 0)
            {
                break;
            }
        }
```

```
}

//which family do we require , AF_INET or AF_INET6
int fm = AF_INET;
struct ifaddrs *ifaddr, *ifa;
int family, s;
char host[NI_MAXHOST];

if (getifaddrs(&ifaddr) == -1)
{
   perror("getifaddrs");
   exit(EXIT_FAILURE);
}

//Walk through linked list, maintaining head pointer so we can free list later
for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next)
{
   if (ifa->ifa_addr == NULL)
   {
      continue;
   }

   family = ifa->ifa_addr->sa_family;

   if (strcmp(ifa->ifa_name, p) == 0)
   {
      if (family == fm)
      {
         s = getnameinfo(ifa->ifa_addr, (family == AF_INET) ? sizeof(struct sockaddr_in) : sizeof(struct sockaddr_in6), host, NI_MAXHOST, NULL, 0, NI_NUMERICHOST);

         if (s != 0)
         {
            printf("getnameinfo() failed: %s\n", gai_strerror(s));
            exit(EXIT_FAILURE);
```

```
            }

            //printf("address: %s", host);
          }
        }
    }
    if (strcmp(host, "") == 0)
    {
       strcat(host, defaultIP);
    }
    strcat(host, " ");
    strcat(host, date);
    strcat(host, " ");
    strcat(host, user);
    strcat(host, " ");
    strcat(host, idp);
    strcat(host, " ");
    strcat(host, comando);
    strcat(host, "\n");
    fputs(host, bitacoraC);
    freeifaddrs(ifaddr);
    //Fin Obtener IP en  variable host

    fclose(bitacoraC);
    return;
}
/*
Autores:
Alumno 01                   Matricula 01
Alumno 02          Matricula 02
Alumno 03                   Matricula 03
Alumno 04                      Matricula 04
Alumno 05          Matricula 05
*/
```

```
/*
registra los errores en
archivo programa_error.log
*/

#include "uamashell.h"

void registraError(char *comando, int pid)
{

    //Variables para abrir archivo bitacora_comandos.log
    FILE *bitacoraC;
    bitacoraC = fopen("../conf/programa_error.log", "at");
    //Obtener usuario
    char *user;
    user = getlogin();
    //Obtener datetime
    char date[20];
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    sprintf(date, "%d-%d-%d %d:%d:%d", tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);
    //PID
    char idp[16];
    sprintf(idp, "Process ID:%d", pid);
    //Obtener IP
    char *defaultIP = "127.0.0.1";
    FILE *f;
    char line[100], *p, *c;

    f = fopen("/proc/net/route", "r");

    while (fgets(line, 100, f))
    {
        p = strtok(line, " \t");
```

```
    c = strtok(NULL, " \t");

    if (p != NULL && c != NULL)
    {
        if (strcmp(c, "00000000") == 0)
        {
            break;
        }
    }
}

//which family do we require , AF_INET or AF_INET6
int fm = AF_INET;
struct ifaddrs *ifaddr, *ifa;
int family, s;
char host[NI_MAXHOST];

if (getifaddrs(&ifaddr) == -1)
{
    perror("getifaddrs");
    exit(EXIT_FAILURE);
}

//Walk through linked list, maintaining head pointer so we can free list later
for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next)
{
    if (ifa->ifa_addr == NULL)
    {
        continue;
    }

    family = ifa->ifa_addr->sa_family;

    if (strcmp(ifa->ifa_name, p) == 0)
    {
```

```
        if (family == fm)
        {
            s = getnameinfo(ifa->ifa_addr, (family == AF_INET) ? sizeof(struct sockaddr_in) : sizeof(struct sockaddr_in6), host, NI_MAXHOST, NULL, 0, NI_NUMERICHOST);

            if (s != 0)
            {
               printf("getnameinfo() failed: %s\n", gai_strerror(s));
               exit(EXIT_FAILURE);
            }

            //printf("address: %s", host);
        }
      }
   }
   if (strcmp(host, "") == 0)
   {
      strcat(host, defaultIP);
   }
   strcat(host, " ");
   strcat(host, date);
   strcat(host, " ");
   strcat(host, user);
   strcat(host, " ");
   strcat(host, idp);
   strcat(host, " ");
   strcat(host, comando);
   strcat(host, "\n");
   fputs(host, bitacoraC);
   freeifaddrs(ifaddr);
   //Fin Obtener IP en  variable host
   fclose(bitacoraC);
   return;
}
/*
```

```
Autores:
Alumno 01                   Matricula 01
Alumno 02          Matricula 02
Alumno 03                   Matricula 03
Alumno 04                      Matricula 04
Alumno 05          Matricula 05
*/

/*
ejecuta los comando exclusivos
del shell
*/

#include "uamashell.h"

int sistema(char **lineacomandos, int donde)
{
	int i =0, contador = 0;
	FILE *file;
	char c;
	//Modificacion para comandos extras
	if (strcmp(*lineacomandos, "terminar") == 0)
	{
		registraCMDS(*lineacomandos, getpid());
		printf("Hasta Luego!\n");

		file = fopen("../conf/instancias.conf", "r");
		while ((c = fgetc(file)) != EOF)
		{
			contador++;
		}
		fclose(file);
		file = fopen("../conf/instancias.conf", "w");
		for (i = 0; i < contador - 1; i++)
		{
```

```
            fputs("x", file);
        }
        fclose(file);
        exit(0);
}
if (strcmp(*lineacomandos, "bitacora_comandos") == 0)
{
        char *archivo = "../conf/programa.log";
        registraCMDS(*lineacomandos, getpid());
        imprimeB(archivo);
        return 0;
}
if (strcmp(*lineacomandos, "bitacora_error") == 0)
{
        char *archivo = "../conf/programa_error.log";
        registraCMDS(*lineacomandos, getpid());
        imprimeB(archivo);
        return 0;
}
if (strcmp(*lineacomandos, "ayuda") == 0)
{
        char *archivo = "../conf/ayuda.log";
        registraCMDS(*lineacomandos, getpid());
        imprimeB(archivo);
        return 0;
}
if (strcmp(*lineacomandos, "configurar") == 0)
{
        char *archivo;
        int i = 0;
        registraCMDS(*lineacomandos, getpid());
        printf("1. Modificar el numero de instancias\n");
        printf("2. Modificar ruta de programa.log\n");
        printf("3. Modificar ruta de programa_error.log\n");
        printf("Indica numero de la configuracion que deseas modificar:");
```

```
        scanf("%d", &i);
        configurar(i);
        return 0;
}
//Fin modificaciones
int estadosalida, retorno;
int pid = fork();

if (pid < 0)
{
        error("Shellcito");
        return (0);
}
if (pid == 0)
{
        char *bin = (char *)malloc(MAXARG * sizeof(char));
        strcat(bin, "/bin/");
        strcat(bin, lineacomandos[0]);
        execv(bin, lineacomandos);
        return (0);
        // registraError(*lineacomandos, pid);
        // error(*lineacomandos);
}
else
{
        /* Código para el padre */
        /* Si es segundo plano imprime el pid y sale */

        if (donde == SEGUNDOPLANO)
        {
                printf("[Process id %i]\n", pid);
                registraCMDS(*lineacomandos, pid);
                return (0);
        }
        else
```

```
            {
                /* Espera hasta que el hijo termine */
                while ((retorno = wait(&estadosalida)) != pid && retorno != -1)
                    ;
                return ((retorno == -1) ? -1 : (estadosalida >>= 8, estadosalida &= 0xFF));
            }
        }
        return (0);
}
/*
Autores:
Alumno 01                   Matricula 01
Alumno 02           Matricula 02
Alumno 03                   Matricula 03
Alumno 04                      Matricula 04
Alumno 05           Matricula 05
*/

/*
main del programa
escribe propmpt y lee la linea
*/

#include "uamashell.h"

int main(int argc, char **argv)
{
        signal(SIGINT, SIG_IGN);
        signal(SIGQUIT, SIG_IGN);

        FILE *inst;
        inst = fopen("../conf/config", "r");
        int N;
        int bandera = 0;
        char c;
```

```
while ((c = fgetc(inst)) != EOF)
{
	if (bandera == 1)
	{
		N = atoi(&c);
		break;
	}
	if (c == 61)
	{
		bandera = 1;
	}
}
int contador = 0;
FILE *file;
file = fopen("../conf/instancias.conf", "r");
while ((c = fgetc(file)) != EOF)
{
	contador++;
}
fclose(file);
if ((contador) <= N - 1)
{
	file = fopen("../conf/instancias.conf", "a");
	fputs("x", file);
	fclose(file);
	char *prompt;
	char bufferentrada[MAXBUF];
	if (argc > 2)
	{
		fprintf(stderr, "Uso:\n%s prompt\n", argv[0]);
		exit(-1);
	}
	prompt = (argc == 1) ? "UAM>" : argv[1];
	while (leerlinea(prompt, bufferentrada) != EOF)
	{
```

```
                procesalinea(bufferentrada);
            }
            return (0);
        }
        else
        {
            printf("Se ha llegado al limite de instancias permitidas\n");
            exit(0);
        }
}
```