Trabajo Práctico Nro.1

Observando el Comportamiento de Linux

SISTEMAS OPERATIVOS I
2013

Esteban Andrés Morales
andresteve07@gmail.com

Facultad de Ciencias Exáctas, Físicas & Naturales, Universidad Nacional de Córdoba, Argentina.

1 de diciembre de 2013

Resumen

En este trabajo práctico se exploró el sistema de archivos virtual /proc de Linux, en busca de información interna del kernel. Con el objetivo de mostrar los resultados de la lectura e interpretación de archivos se programó una aplicación en C denominada ksamp que imprime de diversas formas subconjuntos de la información disponible en /proc.

1. Introduction

Podemos pensar el kernel de Linux como una colección de funciones y estructuras de datos. Estas estructuras de datos (o variables de kernel) contienen la visión del kernel respecto al estado del sistema, donde cada interrupción, cada llamada al sistema, cada fallo de protección hacen que este estado cambie. Inspeccionando las variables

del kernel podemos obtener información relevante a los procesos, interrupciones, dispositivos, sistemas de archivos, capacidades del hardware, etc. Muchas variables conforman el estado del kernel y estas pueden estar alojadas de manera estática o dinámica en un stack frame de la memoria del kernel. Dado que el Kernel de Linux es un programa C de código abierto, es posible inspeccionar el código fuente y encontrar algunos ejemplos relevantes de estas variables. Por ejemplo, en Kernels 2.4.x, xtime definido en `/include/linux/sched.h` que mantiene la hora del sistema, la estructura `task_struct` definida en `/include/linux/sched.h` que contiene la descripción completa de un proceso, o los valores `nr_threads` y `nr_running` definidos en `/kernel/fork.c` los cuales indican cuantos procesos existen y cuantos de estos están corriendo. El código fuente de Linux lo podemos encontrar en el directorio `/usr/src/linux` de la mayoría de las distribuciones, pero también existen páginas donde podemos navegar el código de varias versiones de kernel, como por ejemplo Linux Cross Reference. También existen utilidades para navegación de código local como Source Navigator.

2. Tareas

2.1. Tarea A

Buscar información acerca estructura del directorio `/proc`, y averiguar los siguientes datos, teniendo en cuenta la versión de kernel que está corriendo:

- Tipo y Modelo de CPU.
- Versión del kernel.
- Tiempo de Funcionamiento.
- Uso del Procesador.
- Memoria.
- Solicitudes de Acceso a Disco.
- Cambios de contexto.
- Cantidad de Procesos.

2.2. Tarea B

Se escribió una versión inicial del programa `ksamp` que inspeccione las variables del kernel a través del `/proc` e informe por `stdout`:

- Tipo y Modelo de CPU.
- Versión del kernel.
- Tiempo de Funcionamiento.

Listado 1: Función imprimirB().

```
void imprimirB(){
  cabecera();
  leer_CPU();
  leer_Kernel();
  leer_TiempoT();
}
```

Tipo y Modelo de CPU.

Es posible encontrar la información acerca del tipo y modelo del procesador en el archivo `/proc/cpuinfo`.

Listado 2: Función leer_CPU().

```
void leer_CPU(){

  FILE *f;

  f = fopen("/proc/cpuinfo","r");
  printf("Tipo de CPU:       %s\n",leer_palabra(1,obtener_linea_pos(↩
      f,nro_linea_con_palabra(f,"vendor_id"))));
  printf("Modelo de CPU:     ");
  int i;
  int cant_pal=cant_palabras(obtener_linea_pos(f,↩
      nro_linea_con_palabra(f,"model name")));
  for(i=2;i<cant_pal;i++){
    printf("%s ",leer_palabra(i,obtener_linea_pos(f,↩
        nro_linea_con_palabra(f,"model name"))));
  }
  printf("\n");
  fclose(f);

}
}
```

Versión del kernel.

Puede leerse la versión del Kernel en `/proc/sys/kernel/osrelease`.

Listado 3: Función leer_kernel().

```
void leer_Kernel(){
  FILE *f;

  f = fopen("/proc/sys/kernel/osrelease","r");
  printf("Kernel:            %s",obtener_linea_pos(f,0));
  fclose(f);
}
```

Tiempo de Funcionamiento

Tiempo en días, horas, minutos y segundos que han transurrido desde que se inició el sistema operativo puede encontrarse en `/proc/uptime` y se muestra con el formato `dd:hh:mm:ss`.

Listado 4: Función leer_TiempoT().

```
void leer_TiempoT(){
  FILE *f;

  f = fopen("/proc/uptime","r");
  char* aux = obtener_linea_pos(f,0);
  //printf("_______ %s________\n",aux);
  char* num=leer_palabra(0,aux);
  int segundos=atoi( num );
  int dias, horas, minutos;
    dias = segundos/(3600*24);
    horas = segundos/(60*60);
    segundos %= 60*60;
    minutos = segundos / 60;
    segundos %= 60;
  printf("Up Time:          %i: %i: %i: %i\n",dias,horas,minutos,↩
      segundos);
  fclose(f);
}
```

Cabecera

También se incluye una cabecera donde se indica el nombre de la máquina, la fecha y hora actuales.

Listado 5: Función cabecera().

```
void cabecera(){
  FILE *f;
  f = fopen("/proc/sys/kernel/hostname","r");
  printf("↩
      ************************************************************\↩
      n");
  printf("Nombre:                    %s",obtener_linea_pos(f,0));
  time_t rawtime;
  time ( &rawtime );
  printf("Fecha y hora actual:       %s",ctime(&rawtime));
  printf("↩
      ************************************************************\↩
      n\n");

  fclose(f);
}
```

2.3. Parte C

Se escribe una segunda versión del programa `ksamp` que imprime la misma información que la versión por defecto, pero en caso de invocarse con la opción s, agrega la siguiente información

- Uso del Procesador.
- Cambios de contexto.
- Fecha y Hora de Inicio.
- Cantidad de Procesos.

Listado 6: Función imprimirC().

```
void imprimirC(){
  imprimirB();
  printf("↩
      ____________________________________________________________\↩
      n");
  leer_CPUT();
  leer_CCCONTEXTO();
  FyHInicio();
  leer_NumProc();

}
```

Uso del Procesador

Cuanto tiempo de CPU ha sido empleado para procesos de usuario, de sistema y cuanto tiempo no se usó, para ello se busca en el archivo `/proc/stat`.

Listado 7: Función leer_CPUT().

```
void leer_CPUT(){
  FILE *f;

  f = fopen("/proc/stat","r");
  //char* aux = obtener_linea_pos(f,nro_linea_con_palabra(f,"cpu ")↩
      );
  //printf("_____________ %i___________\n",nro_linea_con_palabra(f,"↩
      cpu"));
  printf("Tiempo de Usuario:         %s USER_HZ (jiffies)\n",↩
      leer_palabra(1,obtener_linea_pos(f,nro_linea_con_palabra(f,"↩
      cpu"))));
  printf("Tiempo de Sistema:         %s USER_HZ (jiffies)\n",↩
      leer_palabra(3,obtener_linea_pos(f,nro_linea_con_palabra(f,"↩
      cpu"))));
  printf("Tiempo sin uso:            %s USER_HZ (jiffies)\n",↩
      leer_palabra(4,obtener_linea_pos(f,nro_linea_con_palabra(f,"↩
      cpu"))));
```

```
  fclose(f);



}
```

Cambios de contexto

Cuantos cambios de contexto han sucedido.

Listado 8: Función leer_CCCONTEXTO().

```
void leer_CCCONTEXTO(){
  FILE *f;
  f = fopen("/proc/stat","r");
  printf("Cambios de Contexto:      %s\n", leer_palabra(1,↩
      obtener_linea_pos(f,nro_linea_con_palabra(f,"ctxt"))));
  fclose(f);
}
```

Fecha y Hora de Inicio

Se especifica la fecha y hora cuando el sistema fue iniciado.

Listado 9: Función FyHInicio().

```
void FyHInicio(){
  FILE *f;
  f=fopen("/proc/uptime","r");

  char* linea=obtener_linea_pos(f,0);
  //printf("Fecha y hora de inicio:   %s",linea);
  char* tiempo_str=leer_palabra(0,linea);

  int tiempo_int=atoi(tiempo_str);

  time_t tiempo_t; // declare a time_t variable
  tiempo_t = (time_t) tiempo_int; // case the time_t variable to a ↩
      long int

  time_t tiempo_actual =time(0);
  time_t tiempo_ini=tiempo_actual-tiempo_t;
  printf("Fecha y hora de inicio:   %s",ctime(&tiempo_ini));
  fclose(f);
}
```

Cantidad de Procesos

Se muestra la cantidad de procesos que se crearon desde que inició el sistema.

Listado 10: Función leer_NumProc().

```
void leer_NumProc(){
  FILE *f;
  f = fopen("/proc/stat","r");
  printf("Nro Procesos:              %s\n", leer_palabra(1,↩
      obtener_linea_pos(f,nro_linea_con_palabra(f,"processes"))));
  fclose(f);
}
```

2.4. ParteD

La tercera parte involucra imprimir todo lo de las versiones anteriores, pero cuando se invoca con la opción `-l tmp int` donde tmp e int son números enteros siendo el primero el tiempo de muestreo y el segundo el intervalo de impresión, además:

- Solicitudes de acceso al Disco.
- Memoria Instalada.
- Memoria Disponible.
- Fecha y Hora de Inicio.
- Lista de promedio de carga por 1 minuto.

Solicitudes de Acceso a Disco

Cuantos pedidos de lectura/escritura a disco se han realizado.

Listado 11: Función leer_NumPetD().

```
void leer_NumPetD(){
  FILE *f;
  f = fopen("/proc/diskstats","r");

  printf("Peticiones a disco:                    %s\n", ↩
      leer_palabra(3,obtener_linea_pos(f,nro_linea_con_palabra(f,"↩
      sda"))));
  fclose(f);
}
```

Memoria Instalada

Cuanta memoria se configuró en el Hardware.

Listado 12: Función leer_TotalMEM().

```
void leer_TotalMEM(){
  FILE *f;
```

```
  f = fopen("/proc/meminfo","r");
  int memt=atoi(leer_palabra(1,obtener_linea_pos(f,0)));
  printf("Memoria Total:                          %i KB\n",memt);
  fclose(f);
}
```

Memoria Disponible

Cuanta memoria tiene y cuanta está disponible.

Listado 13: Función leer_MEMDisp().

```
void leer_MEMDisp(){
  FILE *f;
  f = fopen("/proc/meminfo","r");
  int memd=atoi(leer_palabra(1,obtener_linea_pos(f,1)));
  printf("Memoria Disponible:                     %i KB\n",memd);
  fclose(f);
}
```

Listado de Promedio de Carga

Se muestra el promedio de carga por minuto en la pantalla cuando se hace uso de la opción `-l tmp int` donde tmp e int son números enteros siendo el primero el tiempo de muestreo y el segundo el intervalo de impresión:

Listado 14: Función imprimirD().

```
void imprimirD(const char* p, const char* t){
  imprimirC();
  char* periodo=strdup(p);
  char* totalt=strdup(t);
  int per=atoi(periodo);
  int total=atoi(totalt);

  int i=0;
  while(i<(total/per)){
  printf("↩
      ____________________________________________________________________\↩
      n");
  leer_NumPetD();
  leer_TotalMEM();
  leer_MEMDisp();
  leer_promCar();
  sleep(per);
  i++;
  }
}
```

2.5. Tareas Adicionales

Se implementó la funcionalidad adicional de mostrar los valores que estaban en bytes en MBytes, para acceder a esta función debe usarse la opción `-h` o bien `--humanamente`. A continución se listan las funciones.

Listado 15: Función leer_MEMDispPH().

```
void leer_MEMDispPH(){
  FILE *f;
  f = fopen("/proc/meminfo","r");
  int memd=atoi(leer_palabra(1,obtener_linea_pos(f,1)));
  printf("Memoria Disponible:                    %i MB\n",memd↩
      /1024);
  fclose(f);
}
```

Listado 16: Función leer_TotalMEMPH().

```
void leer_TotalMEMPH(){
  FILE *f;
  f = fopen("/proc/meminfo","r");
  int memt=atoi(leer_palabra(1,obtener_linea_pos(f,0)));
  printf("Memoria Total:                         %i MB\n",memt↩
      /1024);
  fclose(f);
}
```

Listado 17: Función imprimirDPH().

```
void imprimirDPH(const char* p, const char* t){
  imprimirC();
  char* periodo=strdup(p);
  char* totalt=strdup(t);
  int per=atoi(periodo);
  int total=atoi(totalt);

  int i=0;
  while(i<(total/per)){
  printf("↩
      ____________________________________________________________\↩
      n");
  leer_NumPetD();
  leer_TotalMEMPH();
  leer_MEMDispPH();
  leer_promCar();
  sleep(per);
  i++;
  }
}
```

Adicionalemente cabe mencionar que se hizo uso de la función `getopt()` de POSIX.2 para el parseo de comandos. A continuación se lista la función principal de la aplicación que implementa la rutina de parseo:

Listado 18: Función main().

```
int main(int argc, char** argv)
{
  int siguiente_opcion;
  const struct option op_largas[] =
  {
      { "help",          0,  NULL,   'H'},
      { "humanamente",   0,  NULL,   'h'},
      { "incisoC",       0,  NULL,   's'},
      { "incisoD",       1,  NULL,   'l'},
      { NULL,            0,  NULL,   0  }
  };

  const char* tiempo = NULL ;
  const char* per = NULL ;

  int verbose = 0;
  nombre_programa = argv[0];
  if (argc == 1)
  {
      imprimirB();
      exit(EXIT_SUCCESS);
  }
  int funcion_s=0;
  int funcion_l=0;
  int para_humanos=0;
  while (1)
  {
      if(optind>=argc)
      exit(1);
      else {
      char* aux=argv[optind];
      if((pos_caracter_en_cadena('-',aux)!=0) && !funcion_l){
      printf("..::ERROR::..");
      imprime_uso();
      exit(1);}
     }
      siguiente_opcion = getopt_long (argc, argv, op_cortas, ↩
          op_largas, NULL);

      if (siguiente_opcion == -1)
          break;

      switch (siguiente_opcion)
      {
          case 'h' :
              funcion_l=0;
              if(argc==2){
          imprime_uso();
          exit(1);
```

```
}
      para_humanos=1;
      break;
  case 'H' :
      funcion_l=0;
      imprime_uso();
      exit(EXIT_SUCCESS);

  case 's' :
      funcion_s=1;
      funcion_l=0;
      imprimirC();
      break;

  case 'l' :
      {
funcion_l=1;
if(funcion_s == 1){
  imprime_uso();
  exit(1);
}
int i=0;
int pos_l=0;
while( i<argc && pos_l==0 ){
  if(strcmp(argv[i],"-l")==0)
    pos_l=i;
  i++;
}
if(argc!=pos_l+3){
  imprime_uso();
  exit(1);
}
per = optarg;
char* aux2 ;
aux2=strdup(per);
int arg_valido=contiene_solo_nros(aux2);
  if(!arg_valido){
    imprime_uso();
    exit(1);
  }
  else
  {
      tiempo = argv[optind];
      char* aux3 = strdup(tiempo);
      if(!contiene_solo_nros(aux3)){
        imprime_uso();
        exit(1);
      }
      if(para_humanos)
      imprimirDPH(per,tiempo);
      else
      imprimirD(per,tiempo);
  }
}
      break;

  case '?' :
```

```
                funcion_l=0;
                imprime_uso();
                exit(1);

            case -1 :
                funcion_l=0;
                break;

            default :
                abort();
        }
    }
    return 0;

}
```