Trabajo Práctico 2: Primera Entrega

72.11 - Sistemas Operativos

Segundo Cuatrimestre 2024

Integrantes del **Grupo 21**:

62872	Matías LEPORINI
63382	Camila LEE
63074	Ana NEGRE

Instrucciones de Compilación

Para crear el contenedor dentro del directorio que contiene el repositorio:

```
docker pull agodio/itba-so-multi-platform:3.0

cd $MyDir

docker run -v "{$PWD}:/root" --security-opt seccomp:unconfined -ti
agodio/itba-so-multi-platform:3.0
```

Una vez iniciado el contenedor, correr los comandos:

```
cd root

cd TP2-SistemasOperativos-72.11
```

Se provee para la compilación un script que compila el proyecto en su totalidad. El mismo compila tanto el kernel como el módulo de usuario y la toolchain. Para utilizarlo, dentro del contenedor puede correr el comando:

```
bash ./compile.sh
```

Instrucciones de Ejecución

Para ejecutar el proyecto, puede hacer uso del script provisto para tal fin. Fuera del contenedor, debe ejecutar:

```
bash ./run.sh
```

Decisiones de Desarrollo

Para la implementación del primer memory manager se decidió basarse en la implementación de Kernighan & Ritchie obtenida de “The C Programming Language”. La misma está basada en una *linked list* de bloques libres.

Se decidió obviar de la implementación del libro los chequeos de coalición de bloques para disminuir la fragmentación y el uso de *union* para alinear la memoria, ya que no entraba dentro de los requerimientos de la primera entrega parcial.

Para la ejecución del test de memoria, se decidió agregar a la implementación del memory manager la posibilidad de que las aplicaciones de usuario utilicen el

`malloc()` y *`free()`* en forma de ***system calls***. Se modificó la shell de forma tal que el comando para ejecutar el test se encuentre dentro de los comandos aceptados por la misma. De esta forma, para ejecutar el test de memoria una vez inicializado el proyecto, basta con utilizar el comando `testmm` dentro de la interfaz de la shell.

Sin embargo, si se desease ejecutar el test dentro del kernel, basta con descomentar la siguiente porción de código de `main` en `Kernel/kernel.c`, y luego recompilar y re-ejecutar el proyecto:

```
   /*
     To run the memory test in kernel uncomment the following
section:
   */
   /*
   char buffer[200];
     intToStr((size_t) (endHeapAddres - startHeapAddres), buffer,
16);
   char * argv[1] = {buffer};
   uint64_t result = test_mm(1, argv);
   if (result == -1) {
       print("Memory test failed\n");
   }
   */
```

Adicionalmente, se decidió agregar impresiones a la pantalla dentro del test proporcionado con la finalidad de aportar mayor claridad respecto a qué hace la función en cada momento, además de facilitar el *debugging* en caso de error.

Observaciones

En relación al warning lanzado por **gcc** en la compilación, se decidió ignorarlo ya que, al tratarse de un aviso sobre la recursión de la shell, se consideró un despropósito tratarlo. La shell implementada al momento de la primera entrega funciona en un loop infinito ya que en la realización del TPE de Arquitectura de Computadoras no se requería la creación de procesos, falta que será corregida a lo largo de las entregas del trabajo práctico.

Por el otro lado, al correr el *analizer* de **PVS**, se encontraron numerosos *warnings* relacionados a casteos a `void *` , los cuales fueron ignorados, y relacionados a archivos que, o no fueron escritos por nosotros, o bien no eran correctos (repetición de código en `sound.c`, por ejemplo), los cuales también fueron ignorados.

Fragmentos de Código Reutilizado

Al utilizar la implementación de *Kernighan & Ritchie* para la creación del primer memory manager, se tuvo en fuerte consideración la lógica del código provisto en la sección 8.7 del libro:

```
typedef long Align;
union header{
      struct{
      union header *ptr;
      unsigned size;
}s;
Align x;
};
```

```
static Header base;
static Header * freep = NULL;
void * malloc(unsigned nbytes){
Header *p, *prevp;
Header *morecore(unsigned);
unsigned nunits;
nunits = (nbytes+sizeof(Header)-1)sizeof(Header)+1;
if(prevp=freep)==NULL){
base.s.ptr = freep = prevp= &base;
base.s.size = 0;
}
for(p=prevp->s.ptr; ; prevp=p, p=p->s.ptr){
if(p->s.size>=nunits){
if(p->s.size == nunits){
prevp->s.ptr = p->s.ptr;
}
else{
p->s.size -=nunits;
p += p->s.size;
p->s.size = nunits;
}
freep = prevp;
return (void*)(p+1);
}
if(p == freep){
if((p = morecore(nunits)) == NULL)
}
}
```

```
}
```

```
void free(void *ap){
      Header * bp, *p;
      bp = (Header *) ap-1;
      for(p= freep; !(bp>p && bp < p->s.ptr); p = p->s.ptr){
if(p>= p->s.ptr && (bp>p || bp < p->s.ptr)){
break;
}
}
if(bp+bp->s.size == p->s.ptr){
      bp->s.size += p->s.ptr->s.size;
      bp->s.ptr = p->s.ptr->s.ptr;
} else
      bp->s.ptr = p->s.ptr;
if(p+p->s.size == bp){
      p->s.size += bp->s.size;
      p->s.ptr = bp->s.ptr;
} else
      p->s.ptr = bp;
freep = p;
}
```