

1. 개요

1) memsize() 시스템 콜 추가 및 이를 호출하는 쉘 프로그램 구현

- * 호출한 프로세스의 메모리 사용량을 출력하는 memsize() 시스템 호출 구현
- 호출한 프로세스의 memory size를 의미하는 myproc의 sz값을 return하도록 구현
- * memsize() 실행 확인
- memsizetest 쉘을 구현
- memsizetest에서는 memsize()를 실행시키며 호출한 process의 메모리 사용량을 출력을 확인한다.

***** 명세의 실행 결과에서 malloc 전, 후 차이가 2048 바이트가 아닌 이유 *****

malloc(2048)을 실행하면 malloc()의 nbytes 값은 2048이 되고, nunits값은 $(2048+8-1)/8+2$ 을 수행하여 157이 된다. 그 후, for문에서 header의 s.ptr을 따라가며 nunits만큼의 공간이 있는지 검사한다. 하지만 해당 예시처럼 충분한 memory 공간을 찾지 못한 채 freep에 도달하면 morecore()를 실행한다. 인자로 전달된 mallo()의 nunits 즉, morecore()의 nu, 157은 4096보다 작기 때문에 nu의 값은 4096이 된다. header의 size는 Align, 즉 long의 size 8bits이고, sbrk()를 통해 $4096*8=32768B$ 만큼 process의 memory size를 늘린다. sbrk() 내부에서는 growproc()이 sz값을 12288B에서 32768B만큼 늘린 45056B로 업데이트한다. 따라서 2048 bytes가 아닌 4096 bytes가 늘어난 process memory size가 출력된다.

2) trace 시스템 콜 추가 및 이를 호출하는 쉘 프로그램

- * 디버깅할 때 도움이 될 수 있는 trace 시스템 콜 구현
- 추적할 시스템 콜을 지정하는 정수 [mask]를 인자로 받음.
- [mask]의 값에 따라 설정되는 시스템 콜이 다름. [mask]는 설정할 시스템 콜들의 번호만큼 2를 거듭제곱한 값들의 합으로 설정됨. (시스템 콜 번호는 syscall.h에 정의되어 있는 번호를 따름.)
- trace 시스템 콜을 통해 호출한 프로세스와 호출 이후 생성하는 모든 자식 프로세스에 대한 mask를 활성화함.
- 시스템 콜 번호가 mask에 설정되어 있으면, 각 시스템 콜이 리턴될 때 프로세스 아이디, 시스템 콜 이름, 리턴 값이 출력되어야 함. mask의 값을 2진수로 표현하였을 때 값이 1인 자리수가 번호인 시스템 콜을 mask에 설정되었다고 판단함.

- * trace 시스템 콜 실행 확인
- ssu_trace 쉘 프로그램 구현
- _ ssu_trace
- \$ ssu_trace [mask] [command]
- mask는 시스템 콜을 추적하기 위한 비트 집합으로, trace()의 인자로 넘겨주어 프로세스의 mask 값이 됨.
- exec를 통해 [command]를 실행하여, [command]가 호출하는 시스템 콜을 추적

2. 결과

1) memsize() 시스템 콜 추가 및 이를 호출하는 간단한 셸 프로그램 구현

- 프로세스의 메모리 사용량을 출력함
- malloc 사용 후 메모리 사용량을 출력함.

```
kibum@kibum-VirtualBox: ~/xv6-public
kibum@kibum-VirtualBox:~$ cd xv6-public
kibum@kibum-VirtualBox:~/xv6-public$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ memsize
The process is using 12288B
Allocating more memory
The process is using 45056B
Freeing memory
The process is using 45056B
$
```

2) trace 시스템 콜 추가 및 이를 호출하는 간단한 셸 프로그램

- trace의 값을 32, 2130080, 32800, 32928로 설정하여 차례대로 실행한 결과

```
kibum@kibum-VirtualBox: ~/xv6-public
kibum@kibum-VirtualBox:~$ cd xv6-public
kibum@kibum-VirtualBox:~/xv6-public$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ssu_trace 32 grep ssu os README
syscall traced: pid = 3, syscall = read, 1023 returned
syscall traced: pid = 3, syscall = read, 988 returned
syscall traced: pid = 3, syscall = read, 275 returned
syscall traced: pid = 3, syscall = read, 0 returned
$ ssu_trace 2130080 grep ssu os README
syscall traced: pid = 4, syscall = exec, 0 returned
syscall traced: pid = 4, syscall = open, 3 returned
syscall traced: pid = 4, syscall = read, 1023 returned
syscall traced: pid = 4, syscall = read, 988 returned
syscall traced: pid = 4, syscall = read, 275 returned
syscall traced: pid = 4, syscall = read, 0 returned
syscall traced: pid = 4, syscall = close, 0 returned
$ ssu_trace 32800 grep ssu os README
syscall traced: pid = 5, syscall = open, 3 returned
syscall traced: pid = 5, syscall = read, 1023 returned
syscall traced: pid = 5, syscall = read, 988 returned
syscall traced: pid = 5, syscall = read, 275 returned
syscall traced: pid = 5, syscall = read, 0 returned
$ ssu_trace 32928 grep ssu os README
syscall traced: pid = 6, syscall = exec, 0 returned
syscall traced: pid = 6, syscall = open, 3 returned
syscall traced: pid = 6, syscall = read, 1023 returned
syscall traced: pid = 6, syscall = read, 988 returned
syscall traced: pid = 6, syscall = read, 275 returned
syscall traced: pid = 6, syscall = read, 0 returned
$
```

3. 소스코드

1) sysproc.c

- sys_memsize() 추가
- sys_memsize()는 process의 memory size를 의미하는 myproc()->sz를 출력한다.
- sys_trace() 추가
- sys_trace()는 인자로 받은 integer형의 값을 myproc()->mask에 전달하고, 오류가 발생하면 -1을 return 한다.

```
93 int
94 sys_memsize(void)
95 {
96     uint size;
97
98     size = myproc()->sz;
99
100    return size;
101 }
102
103 int
104 sys_trace(void)
105 {
106     if(argint(0, &myproc()->mask) < 0)
107         return -1;
108
109     return 0;
110 }
```

2) syscall.h

- SYS_memsize, SYS_trace 등록

```
1 // System call numbers
2 #define SYS_fork    1
3 #define SYS_exit    2
4 #define SYS_wait    3
5 #define SYS_pipe    4
6 #define SYS_read    5
7 #define SYS_kill    6
8 #define SYS_exec    7
9 #define SYS_fstat   8
10 #define SYS_chdir   9
11 #define SYS_dup    10
12 #define SYS_getpid  11
13 #define SYS_sbrk    12
14 #define SYS_sleep   13
15 #define SYS_uptime  14
16 #define SYS_open    15
17 #define SYS_write   16
18 #define SYS_mknod   17
19 #define SYS_unlink  18
20 #define SYS_link    19
21 #define SYS_mkdir   20
22 #define SYS_close   21
23 #define SYS_memsize 23
24 #define SYS_trace   24
```

3) proc.h

- trace mask를 의미하는 mask 변수 추가

```
37 // Per-process state
38 struct proc {
39     uint sz;                // Size of process memory (bytes)
40     pde_t* pgdir;          // Page table
41     char *kstack;          // Bottom of kernel stack for this process
42     enum procstate state;   // Process state
43     int pid;               // Process ID
44     struct proc *parent;    // Parent process
45     struct trapframe *tf;   // Trap frame for current syscall
46     struct context *context; // swtch() here to run process
47     void *chan;            // If non-zero, sleeping on chan
48     int killed;            // If non-zero, have been killed
49     struct file *ofile[NOFILE]; // Open files
50     struct inode *cwd;      // Current directory
51     char name[16];         // Process name (debugging)
52     int mask;              // Trace mask
53 };
```

4) syscall.c

- sys_memsizes, sys_trace 등록

```
85 extern int sys_chdir(void); 109 static int (*syscalls[])(void) = {
86 extern int sys_close(void); 110 [SYS_fork] sys_fork,
87 extern int sys_dup(void); 111 [SYS_exit] sys_exit,
88 extern int sys_exec(void); 112 [SYS_wait] sys_wait,
89 extern int sys_exit(void); 113 [SYS_pipe] sys_pipe,
90 extern int sys_fork(void); 114 [SYS_read] sys_read,
91 extern int sys_fstat(void); 115 [SYS_kill] sys_kill,
92 extern int sys_getpid(void); 116 [SYS_exec] sys_exec,
93 extern int sys_kill(void); 117 [SYS_fstat] sys_fstat,
94 extern int sys_link(void); 118 [SYS_chdir] sys_chdir,
95 extern int sys_mkdir(void); 119 [SYS_dup] sys_dup,
96 extern int sys_mknod(void); 120 [SYS_getpid] sys_getpid,
97 extern int sys_open(void); 121 [SYS_sbrk] sys_sbrk,
98 extern int sys_pipe(void); 122 [SYS_sleep] sys_sleep,
99 extern int sys_read(void); 123 [SYS_uptime] sys_uptime,
100 extern int sys_sbrk(void); 124 [SYS_open] sys_open,
101 extern int sys_sleep(void); 125 [SYS_write] sys_write,
102 extern int sys_unlink(void); 126 [SYS_mknod] sys_mknod,
103 extern int sys_wait(void); 127 [SYS_unlink] sys_unlink,
104 extern int sys_write(void); 128 [SYS_link] sys_link,
105 extern int sys_uptime(void); 129 [SYS_mkdir] sys_mkdir,
106 extern int sys_memsizes(void); 130 [SYS_close] sys_close,
107 extern int sys_trace(void); 131 [SYS_memsizes] sys_memsizes,
132 [SYS_trace] sys_trace,
133 };
```

- 시스템 콜 이름 배열 구현

```
135 char *syscall_name[] = {
136 [SYS_fork] "fork",
137 [SYS_exit] "exit",
138 [SYS_wait] "wait",
139 [SYS_pipe] "pipe",
140 [SYS_read] "read",
141 [SYS_kill] "kill",
142 [SYS_exec] "exec",
143 [SYS_fstat] "fstat",
144 [SYS_chdir] "chdir",
145 [SYS_dup] "dup",
146 [SYS_getpid] "getpid",
147 [SYS_sbrk] "sbrk",
148 [SYS_sleep] "sleep",
149 [SYS_uptime] "uptime",
150 [SYS_open] "open",
151 [SYS_write] "write",
152 [SYS_mknod] "mknod",
153 [SYS_unlink] "unlink",
154 [SYS_link] "link",
155 [SYS_mkdir] "mkdir",
156 [SYS_close] "close",
157 [SYS_memsizes] "memsize",
158 [SYS_trace] "trace",
159 };
```

- syscall() 수정

- 만약 mask를 num 만큼 왼쪽으로 shift한 값을 2로 나눈 값이 1이면, mask를 이진수로 표현했을 때, mask의 num 번째 숫자가 1이므로, 해당 num의 syscall은 mask로 설정된 것으로 판단한다.
- 해당 시스템 콜이 리턴되면 프로세스 아이디, 이름, 리턴 값을 출력한다.

```
162 void
163 syscall(void)
164 {
165     int num;
166     struct proc *curproc = myproc();
167
168     num = curproc->tf->eax;
169     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
170         curproc->tf->eax = syscalls[num]();
171         if((curproc->mask >> num) % 2 == 1){
172             cprintf("syscall traced: pid = %d, syscall = %s, %d returned\n",
173                 curproc->pid, syscall_name[num], curproc->tf->eax);
174         }
175     } else {
176         cprintf("%d %s: unknown sys call %d\n",
177             curproc->pid, curproc->name, num);
178         curproc->tf->eax = -1;
179     }
180 }
```

5) user.h

- memsize, trace 등록

```
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int memsize(void);
27 int trace(int);
```

6) usys.S

- memsize, trace 등록

```
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(memsize)
33 SYSCALL(trace)
```

7) proc.c

- fork() 수정
- 'np->mask = curproc->mask;'를 추가하여 자식 프로세스의 mask를 활성화한다.

```
200 np->sz = curproc->sz;
201 np->parent = curproc;
202 *np->tf = *curproc->tf;
203 np->mask = curproc->mask;
```

- allocproc() 수정
- 마스크 값을 0으로 초기화해줌.

```
88 found:
89 p->state = EMBRYO;
90 p->pid = nextpid++;
91 p->mask = 0;
```


8) memsizetest.c

- memsize()를 통해 process의 memory size를 출력한다.
- memsize()를 통해 malloc() 사용 후, 늘어난 memory size를 확인한다.

```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4
5 #define SIZE 2048
6
7 int main(void)
8 {
9     int msize = memsize();
10    printf(1, "The process is using %dB\n", msize);
11
12    char *tmp = (char*)malloc(SIZE* sizeof(char));
13
14    printf(1, "Allocating more memory\n");
15    msize = memsize();
16    printf(1, "The process is using %dB\n", msize);
17
18    free(tmp);
19    printf(1, "Freeing memory\n");
20    msize = memsize();
21    printf(1, "The process is using %dB\n", msize);
22
23    exit();
24 }
```

9) ssu_trace.c

- 첫 번째로 받은 인자, mask값을 trace로 전달하여 process의 mask값을 할당한다.
- 첫 번째 이후의 인자를 command로 하여 exec를 통해 실행한다.

```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4
5 int main(int argc, char *argv[])
6 {
7     int mask = atoi(argv[1]);
8     trace(mask);
9
10    char *argv2[argc-2];
11
12    for(int i=0; i<argc-1; i++){
13        argv2[i] = argv[i+2];
14    }
15
16    exec(argv[2], argv2);
17
18    exit();
19 }
```

10) Makefile

- memsizetest, ssu_trace 추가

```
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _zombie\
184     _memsizetest\
185     _ssu_trace
```

```
252 EXTRA=\
253     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
254     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
255     printf.c umalloc.c memsizetest.c ssu_trace.c\
256     README list.txt dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
257     .gdbinit.tmpl gdbutil\
```