Lab1 Report

Yingfan Gao; Yufei Li; Shahab Nikkhoo

Demo Link

1 The list of all files modified

• kernel/syscall.h: Define new system call number
• kernel/syscall.c: Updata system call table
• kernel/sysproc.c: Define system call function
• kernel/proc.c: Create new kernel function
• kernel/defs.h: Create new system call
• user/usys.pl and user/user.h: Update user-space system call interface
• user/test.c: Write a user program
• kernel/kalloc.c: Allocate physical memory for process
• kernel/proc.h: Define data structure for process

```
23 #define SYS_hello  22
24 #define SYS_sysinfo 23 //lab1
25 #define SYS_procinfo 24
26
```

Figure 1: syscall.h

```
104 extern uint64 sys_hello(void);
105 extern uint64 sys_info(void);
106 extern uint64 sys_procinfo(void);
107
108
109 // An array mapping syscall numbers from sys
110 // to the function that handles the system c
111 static uint64 (*syscalls[])(void) = {
112 [SYS_fork]    sys_fork,
113 [SYS_exit]    sys_exit,
114 [SYS_wait]    sys_wait,
115 [SYS_pipe]    sys_pipe,
116 [SYS_read]    sys_read,
117 [SYS_kill]    sys_kill,
118 [SYS_exec]    sys_exec,
119 [SYS_fstat]   sys_fstat,
120 [SYS_chdir]   sys_chdir,
121 [SYS_dup]     sys_dup,
122 [SYS_getpid]  sys_getpid,
123 [SYS_sbrk]    sys_sbrk,
124 [SYS_sleep]   sys_sleep,
125 [SYS_uptime]  sys_uptime,
126 [SYS_open]    sys_open,
127 [SYS_write]   sys_write,
128 [SYS_mknod]   sys_mknod,
129 [SYS_unlink]  sys_unlink,
130 [SYS_link]    sys_link,
131 [SYS_mkdir]   sys_mkdir,
132 [SYS_close]   sys_close,
133 [SYS_hello]   sys_hello,
134 [SYS_sysinfo] sys_info,
135 [SYS_procinfo] sys_procinfo,
136 };
```

Figure 2: syscall.c

2 Explanation on Change

2.1 Part1

Similar to the demo which was presented in class, we first defined new system call number 23 to the new created system call [SYS_sysinfo] and then updated system call table (as shown in Figure 1 and Figure 2)

Next, we defined system call function [sysinfo(int param)] and program it in "kernel/proc.c" (as shown in Figure 3 and Figure 4) Also, we add this function into "kernel/defs.h" that enable the new created system call available. Then, we update user-space system call interface with the new system call (as shown in Figure 5, Figure 6 and Figure 7)

Furthermore, we define the function "free_memory_pages" by utilizing kmem in "kernel/kalloc.c" to complete the taks3 in part1. (as shown in Figure 8)

2.2 Part2

Most of the parts in the change of part2 is similar to those in the part1, except the pro-

```c
uint64
sys_info(void)
{
  int param;
  argint(0, &param);

  return sysinfo(param);
}

uint64
sys_procinfo(void)
{
  uint64 addr;

  argaddr(0, &addr);
  struct pinfo* pi = (struct pinfo*)addr;

  return procinfo(pi);
}
```

Figure 3: sysproc.c

```c
int sysinfo(int param)
{
  struct proc * p = myproc();// struct proc * p = myproc();
  int number_result = 0; //initialize

  if (param == 0)
  {
    for(p = proc; p < &proc[NPROC]; p++)
    {
      if(p->state != UNUSED)
      {
        number_result++;
      }
    }
    printf("Total number of active processes in the system:");
    return number_result;
  }

  else if (param == 1)
  {
    printf("Total number of system calls since boot up:");
    return systemcall_count[p->pid];
  }

  else if (param == 2)
  {
    printf("The number of free memory pages in the system:");
    return free_memory_pages();
  }
  return -1;
}
```

Figure 4: proc.c_part1

```c
11  void            hello(void);
12  int             sysinfo(int);
13  int             procinfo(struct pinfo*);
```

Figure 5: defs.h

```c
40  entry("sysinfo");
41  entry("procinfo");
42
```

Figure 6: usys.pl

```c
27  int sysinfo(int);
28  int procinfo(struct pinfo*);
```

Figure 7: user.h

```c
int free_memory_pages(void)
{
  int Num_pages = 0;
  struct run * r = kmem.freelist;
  while (r != NULL)
  {
    Num_pages++;
    r = r->next;
  }
  return Num_pages;
}
```

Figure 8: kalloc.c

gramming part in "kernel/proc.c" (as shown in Figure 9). In part2, we need to add a new system call int procinfo(struct pinfo*) that provides information specific to the current process. It takes as input a pointer of [struct pinfo] and fills out the fields of this struct. (as shown in Figure 10).

```c
int procinfo(struct pinfo *in)
{
  struct proc * p = myproc();

  int n = p->parent->pid;
  copyout(p->pagetable, (uint64)&in->ppid, (char *)&n, sizeof(n));

  int m = systemcall_count[p->pid];
  copyout(p->pagetable, (uint64)&in->syscall_count, (char *)&m, sizeof(m));

  uint64 mem_size = p->sz; //memory size of process
  int mem_page = (PGROUNDUP(mem_size))/PGSIZE; // page size

  copyout(p->pagetable, (uint64)&in->page_usage, (char *)&mem_page, sizeof(mem_page));

  return 0;
}
```

Figure 9: proc.c_part2

```c
struct pinfo
{
  int ppid;
  int syscall_count;
  int page_usage;
};
```

Figure 10: pinfo

The test of implementation was completed exactly by using the user-level program provided by the example (as shown in Figure 11) This user program calls sysinfo() and creates N child processes, each of which allocates a specified amount of memory and calls procinfo(). Once all child processes print procinfo(), the parent process calls sysinfo() again.

Figure 11: test

3 Detailed description of XV6 source code

In general, we defined a new system call [sys_sysinfo], which calls the kernel function [sysinfo] to collect parameters offerby by the user to complete the desired tasks. According to the received parameter as input, this call will return:

•If param == 0: the total number of active processes (ready, running, waiting, or zombie) in the system.

•If param == 1: the total number of system calls that has made so far since the system boot up.

•If param == 2: the number of free memory pages in the system.

For the first task (as shown in Figure 12), firstly, we initialized the total number of process to 0. Then we run a loop until the number of processes in the system equals to maximum. The condition was set as whether the state of process is UNUSED. Once the state is not UNUSED, the value is increased by 1. Finally, the result number is returned.

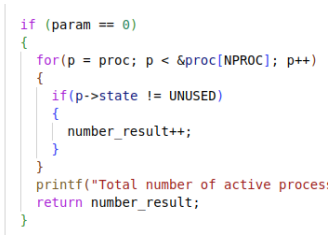

Figure 12: Number of active process

In the second task (as shown in Figure 13), we defined [uint systemcallcount] in "kernel/syscall.c" and added it into [syscall(void)], using cumulative calculation to get he total number of system calls that has made so far since the system boot up. For the interface between system call and its function, we had to declare [systemcallcount] in "kernel/defs.h" again. What's more, it was worth mentioning that in order to exclude the current [sysinfo] syscall's attempt when returning the result, 1 had to be subtracted from the final value.

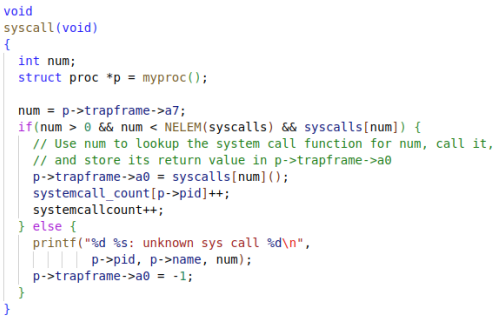

Figure 13: Number of system calls

In the final task of part1 (as shown in Figure 8), one integer [Num_pages] was initialized in advance. To return the number of free memory pages available, we first declare a pointer to a [struct] called [r] that is set to the [freelist] member of an object called [kmem]. The function then enters a while loop where it checks if the pointer is not NULL, meaning there's still memory blocks available. Inside the while loop, it increments the [Num_pages]

variable by one and assigns to [r] the next member of the current [struct] it's pointing to.

The loop continues until the pointer becomes NULL, indicating the end of the list of free memory blocks. Then the function returns the [Num_pages] variable.

```
int procinfo(struct pinfo *in)
{
  struct proc * p = myproc();

  int n = p->parent->pid;
  copyout(p->pagetable, (uint64)&in->ppid, (char *)&n, sizeof(n));

  int m = systemcall_count[p->pid];
  copyout(p->pagetable, (uint64)&in->syscall_count, (char *)&m, sizeof(m));

  uint64 mem_size = p->sz; //memory size of process
  int mem_page = (PGROUNDUP(mem_size))/PGSIZE; // page size

  copyout(p->pagetable, (uint64)&in->page_usage, (char *)&mem_page, sizeof(mem_page));

  return 0;
}
```

Figure 14: proc.c_part2

After defining and declaring essential variables in files from both kernel and user folders, we wrote a function [procinfo] in "kernel/proc.c"(s shown in Figure 14). Before explaining this function, the function [syscall] in "kernel/syscall.c" had to be mentioned again. To implement this function, we declared an array named [systemcall_count] with the biggest size in the system (NPROC). When a process is created, the function [allocproc] in the "kernel/proc.h" is used to initialize the process and set the array to 0. When a process is destroied, the function [freeproc] in the same file is used to free the resources and also set that array to zero so that it can be used by a new process later. In the file "kernel/syscall.c", the number of system call is increased by 1 for each corresponding process ID. This allows us to obtain the total number of system calls made by a process by simply providing its process ID. At last, by accessing the systemcall_count array, we got the desired result as total number of system calls made by the current process.

The function named [procinfo] in "kernel/proc.c" took a pointer to a struct called [pinfo] as an input. It started by declaring a pointer to a struct called [p] that is set to the return value of function called [myproc()], which returned a pointer to a struct that represents the current process.

Next, it declared an integer variable [n] and assigned it to the value of the [pid] member of the parent process of the current process. The function then used a function called [copyout] to copy the value of [n] to the address of the [ppid] member of the [pinfo] struct, which is passed as an argument to the function.

Later, it declared an integer variable [m] and assigned it to the value of the number of system calls made by the current process, which is accessed through "systemcall_count" mentioned above using the [pid] of the current process as an index. The "copyout" was used again to copy the value of [m] to the address of the [syscall_count] member of the [pinfo].

Finally, it declared an integer variable "mem_page" and assigned it to the number of pages used by the current process's memory size, which is accessed through the [sz] member of the [proc] struct, and it used [PGROUNDUP] and [PGSIZE] to round up the memory size to the nearest page size. The value of [mem_page] to the address of the [page_usage] member of the [pinfo] struct was copied as well.

4 Summary of the contribution

In this lab assignment, Yufei took charge of the first part of the lab, while Shahab completed the second part. And Yingfan was responsible for the writting of lab report.