Homework 04 Report

1. syscall.h

```
diff --git a/syscall.h b/syscall.h
index bc5f356..0907111 100644
--- a/syscall.h
+++ b/syscall.h
@@ -20,3 +20,5 @@
 #define SYS_link   19
 #define SYS_mkdir  20
 #define SYS_close  21
+#define SYS_date   22
+#define SYS_ps     23
root@da07666c03f5:/workspaces/xv6-public#
```

   add SYS call for date and ps;

2. usys.S

```
root@da07666c03f5:/workspaces/xv6-public# git diff usys.S
diff --git a/usys.S b/usys.S
index 8bfd8a1..6fd8c07 100644
--- a/usys.S
+++ b/usys.S
@@ -29,3 +29,7 @@ SYSCALL(getpid)
 SYSCALL(sbrk)
 SYSCALL(sleep)
 SYSCALL(uptime)
+SYSCALL(date)
+SYSCALL(ps)
```

   register SYS call for date and ps; thus the macro can parse the syscall;

3. syscall.c

```
@@ -103,6 +103,8 @@ extern int sys_unlink(void);
 extern int sys_wait(void);
 extern int sys_write(void);
 extern int sys_uptime(void);
+extern int sys_date(void);
+extern int sys_ps(void);

 static int (*syscalls[])(void) = {
 [SYS_fork]    sys_fork,
@@ -126,6 +128,9 @@ static int (*syscalls[])(void) = {
 [SYS_link]    sys_link,
 [SYS_mkdir]   sys_mkdir,
 [SYS_close]   sys_close,
+[SYS_date]    sys_date,
+[SYS_ps]      sys_ps,
+
 };
```

   declare syscall, and add syscall function pointer to syscall array;

4. sysproc.c

```
+
+int
+sys_date(void)
+{
+  struct rtcdate *date;
+  if (argptr(0, (void*)&date, sizeof(*date)) < 0) {
+      return -1;
+  }
+  cmostime(date);
+  return 0;
+}
+
+int
+sys_ps(void)
+{
+  return ps();
+}
```

add implementation for sys_date and sys_ps to kernel source;

5. user.h

```
root@da07666c03f5:/workspaces/xv6-public# git diff user.h
diff --git a/user.h b/user.h
index 4f99c52..68793e0 100644
--- a/user.h
+++ b/user.h
@@ -23,6 +23,8 @@ int getpid(void);
 char* sbrk(int);
 int sleep(int);
 int uptime(void);
+int date(struct rtcdate *);
+int ps(void);

 // ulib.c
 int stat(const char*, struct stat*);
root@da07666c03f5:/workspaces/xv6-public#
```

expose user-level syscall;

6. date.c

```c
#include "types.h"
#include "user.h"
#include "date.h"

void
display()
{
    struct rtcdate r;
    if (date(&r))
    {
        printf(2, "date failed\n");
        exit();
    }
    printf(1,"%d",r.month);
    printf(1,"/%d",r.day);
    printf(1,"/%d",r.year);
    printf(1," ");
    printf(1,"%d",r.hour);
    printf(1,":%d",r.minute);
    printf(1,":%d\n",r.second);
}

int
main(int argc, char **argv)
{
    if (argc == 1)
        display();

    exit(); //don't use return
}
```

execution for date:

```
$ date
3/13/2023 13:48:59
```

7. proc.c

```
diff --git a/proc.c b/proc.c
index 806b1b1..b2090eb 100644
--- a/proc.c
+++ b/proc.c
@@ -532,3 +532,28 @@ procdump(void)
     cprintf("\n");
   }
 }
+
+int
+ps()
+{
+  struct proc *p;
+  //Enables interrupts on this processor.
+  sti();
+
+  //Loop over process table looking for process with pid.
+  acquire(&ptable.lock);
+  cprintf("name \t pid \t ppid \t\t\t state \n");
+  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
+    if(p->state == SLEEPING)
+      cprintf("%s \t %d \t %d \t\t\t SLEEPING \n ", p->name, p->pid, p->parent->pid);
+    else if(p->state == RUNNING)
+      cprintf("%s \t %d \t %d \t\t\t RUNNING \n ", p->name, p->pid, p->parent->pid);
+    else if(p->state == RUNNABLE)
+      cprintf("%s \t %d \t %d \t\t\t RUNNABLE \n ", p->name, p->pid, p->parent->pid);
+  }
+
+  cprintf("\n\n");
+  release(&ptable.lock);
+
+  return 23;
+}
```

extract info from page table(ptable), and print out;

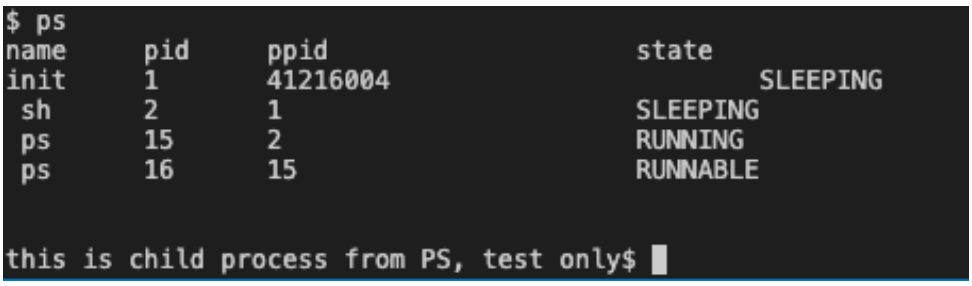
8. ps.c

C ps.c > main(int, char **)

```c
#include "types.h"
#include "user.h"
#include "fcntl.h"
#include "stat.h"

int
main(int argc, char **argv)
{
    int pid = fork();
    if (pid < 0) {
        exit();
    }
    else if (pid > 0) {
        ps();
    }
    else {
        printf(1, "this is child process from PS, TEST ONLY \n");
    }
    wait();
    exit();
}
```

ps execution:



Explaination:

the init process (pid=1) comes with the xv6 OS;

the sh process(pid=1, ppid=1) is forked from init since its ppid points to init;

the ps(pid=15) is forked from sh(pid=2), and it is running as the sh displayed;

the ps(pid=16) is forked from ps(pid=15), since it is from the parent ps, so its state is runnable;