Implement `countTraps` system call

Overview

The purpose of this part is to add the `countTrap()` system call in xv6 to get deep understanding of the control mechanism of an operating system. Here I describe each file that I changed to implement `countTraps()`

Implementation

1. In `user.h` add `countTraps` system call.

```
char* sbrk(int);
int sleep(int);
int uptime(void);
int countTraps(int);
```

2. adding `SYSCALL(countTraps)` in `usys.S`

```
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(countTraps)
```

3. in `syscall.h` adding system call index

```
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_countTraps  22
```

4. adding system call function projection

```
extern int sys_uptime(void);
extern int sys_countTraps(void);
static int (*syscalls[])(void) = {
...
[SYS_close]     sys_close,
[SYS_countTraps]  sys_countTraps
};
```

5. implement `countTraps` system call in kernel

The system call takes one argument, in integer "mask", whose bits specify which system calls to trace. For example, to trace the read system call, a program calls trace(1 << SYS_read), where SYS_read is a system call number from `syscall.h` . If the system call number is set in the mask, we can count how many times this system call execution. Since every time a user program asks

for an operating system service, it will trap into the OS. So by adding all the system call execution times, we could know count the total traps of a user program.

1) add mask and system call traps number to `struct proc` in `proc.h`

```
struct proc {
  ...
  int tracemask;               // trace mask to print syscall
  int syscall_traps;           // get total traps caused by system call;
  int syscall_trap_count[23];  // recored the occurance of every system call;

};
```

2) initialize the `syscall_traps` and `syscall_trap_count` in `proc.c`

```
static struct proc*
allocproc(void)
{
  ...
  p->syscall_traps = 0;
  for(int i = 0; i < 23; i++) {
    p->syscall_trap_count[i] = 0;
  }
  return p;
}
```

3) modify the child process, make it inheritance the mask in `proc.c`

```
int
fork(void)
{
  ...
  np->state = RUNNABLE;
  np->tracemask = curproc->tracemask;
  release(&ptable.lock);
  return pid;
}
```

4) in `sysproc.c` add `sys_countTraps`

```
int
sys_countTraps(void)
{
  int mask;
  if(argint(0, &mask) < 0) {
    return -1;
  }
  myproc()->tracemask = mask;
  return 0;
}
```

5) add projection from system number index to name in `syscall.c`

```
static char *syscallsname[] = {
[SYS_fork]      "fork",
[SYS_exit]      "exit",
[SYS_wait]      "wait",
[SYS_pipe]      "pipe",
[SYS_read]      "read",
[SYS_kill]      "kill",
[SYS_exec]      "exec",
[SYS_fstat]     "fstat",
[SYS_chdir]     "chdir",
[SYS_dup]       "dup",
[SYS_getpid]    "getpid",
[SYS_sbrk]      "sbrk",
[SYS_sleep]     "sleep",
[SYS_uptime]    "uptime",
[SYS_open]      "open",
[SYS_write]     "write",
[SYS_mknod]     "mknod",
[SYS_unlink]    "unlink",
[SYS_link]      "link",
[SYS_mkdir]     "mkdir",
[SYS_close]     "close",
[SYS_countTraps]  "countTraps",
};
```

6) modify `syscall` function in `syscall.c`

```
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();
  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
    if(curproc->tracemask & (1 << num)){
      cprintf("%d: syscall %s -> executes %d times.\n", curproc->pid,
syscallsname[num], ++curproc->syscall_trap_count[num]);
      cprintf("%d: total traps -> %d\n", curproc->pid, ++curproc->syscall_traps);
    }
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

6. add `countTrap` function for user call

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

/*test how many system calls would be executed when the system runs countTraps*/
void test01() {
    countTraps(0);
}

/*test different system calls, taking write for example*/
void test02() {
    printf(3, "Hello!\n");
    countTraps(0);

}

/*test software interrupts, using division by zero as an example*/
void test03() {
    if(fork() == 0) {
        printf(3, "this is a child process.\n");
        int a = 1/0;
        printf(2, "a = %d\n", a);
        exit();
    }
    wait();
    countTraps(0);
}

/*test the parent and the child process, using fork as an example*/
void test04() {
    if(fork() == 0) {
        printf(3, "this is a child process.\n");
        exec("echo", "A");
        countTraps(0);
        exit();
    }
    wait();
    printf(3, "this is a parent process.\n");
    countTraps(0);
}

/*test hardware interrupts, using sleep to mimic*/
void test05() {
    for(int i = 0; i< 10; i++) {
        sleep(1);
        uptime();
    }
    countTraps(0);
}

/*test hardware interrupts, using sleep to mimic*/
void test06() {
    for(int i = 0; i< 10; i++) {
        sleep(1);
        uptime();
```

```
    }
    countTraps(0);
}


int main(int argc, char *argv[]) {
    test01();
    test02();
    test03();
    test04();
    test06();
    exit();
}
```

7. add `_countTraps` to UPROGS in Makefile

```
UPROGS=\
    ...
    _countTraps\

EXTRA=\
    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
    ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
    printf.c countTraps.c umalloc.c\
    ...
```