CS471 ASST2

System Calls

Save all of us some time and do NOT attempt to find a solution online.

Deliverables

- Code walk-through (20 points)
    - Same as previous assignments.
- Implementations (60 points)
    - System calls
        - getpid
        - execv
        - fork
        - waitpid
        - exit
    - Multi-level queue scheduler

Deliverables

- Design document (20 points)
  - Answers to the code walk-through questions.
  - A high level description of how you are approaching the problem. (4 pts)
  - A detailed description of the implementation (e.g., new structures, why they were created, what they are encapsulating, what problems they solve). (6 pts)
  - A discussion of the pros and cons of your approach. (6 pts)
  - Alternatives you considered and why you discarded them. (4 pts)
- The output of the tests.

Extra credit

- Extra Credit: (TBD points)
  - File system calls
    - open
    - read
    - write
    - lseek
    - close
    - dup2
    - chdir
    - getcwd

Configuration

- You can use build-asst2.php file
  - wget mason.gmu.edu/~aroy6/build-asst2.php

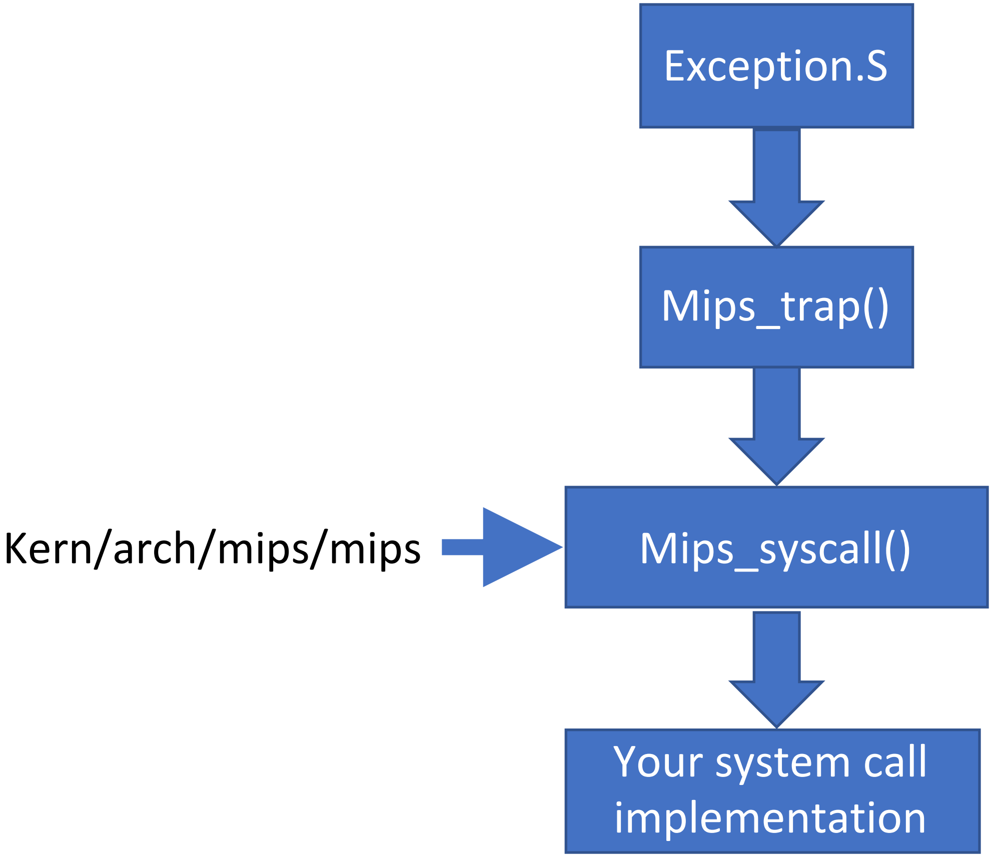
What is a system call

- A System Call is a software interface
  - Part of kernel
  - Called by User-programs
- Why do we need this?
  - The Operating System needs to look over the user programs
    - The “Government” metaphor
  - User-programs have limited privilege (can be erroneous or malicious)
    - Should not be able to access critical resources (e.g., file system) directly
  - Invoking a system call hands control over to the OS, which can execute privileged instructions

User-Level Interface

- Os161-1.11/include/uninstd.h contains the user-level system call interfaces.
  - int execv(const char *prog, char *const *args);
  - pid_t fork(void);
  - int waitpid(pid_t pid, int *returncode, int flags);
  - int open(const char *filename, int flags, ...);
  - int read(int filehandle, void *buf, size_t size);
  - int write(int filehandle, const void *buf, size_t size);
  - int close(int filehandle);
  - int reboot(int code);
  - int sync(void);

How is it linked?



Exception.S

Mips_trap()

Kern/arch/mips/mips

Mips_syscall()

Your system call implementation

Where to put your system call implementation?

- This time no skeleton code is given.
- Create under kern/userprog
  - fork.c
  - execv.c
  - waitpid.c
  - getpid.c
  - exit.c
- Name your system calls sys_{getpid|fork|execv|waitpid|exit}
- Add the new file to kern/conf
  - File userprog/getpid.c
  - The same way you have done hello.c in ASST0.
- Include your system call function definition in kern/include/syscall.h

Process structure

- A common hack.
  - Add the necessary fields to the thread structure and treat it as a process.
    - Pid
    - Exit status
    - Parent process.
    - Etc.

For each system call

- Make sure to increment process counter.
    - Otherwise, it will restart the same system call.
    - Tf->tf_epc+=4
- If error
    - Store the error code in tf->tf_v0
    - Set tf->tf_a3 to 1.
- If no error
    - Store the return value in tf->tf_v0
    - Set tf->tf_a3 to 0.

Sys_getpid

- Simplest one.
- Just return the pid of the executing process.
- Getpid does not fail.

Sys_execv

- Replace the currently executing program image with a new process image.
- Process id is unchanged.
- int sys_execv(char *program, char **args)
  - program: path name of the program to run.
  - Args: tf->tf_a0 and tf->tf_a1
- Most of the implementation is already in the runprogram.c. Add the followings:
  - Check the last argument in **args is NULL.
  - Make sure it is less than MAX_ARGS_NUM
  - *copyin* the arguments from user space to kernel space.
  - Create a new address space.
    - *as_create()*
  - Allocate a stack on it.
    - *as_define_stack()*
  - *Copyout* the arguments back onto the new stack

Sys_execv errors

ENODEV	The device prefix of *program* did not exist.
ENOTDIR	A non-final component of *program* was not a directory.
ENOENT	*program* did not exist.
EISDIR	*program* is a directory.
ENOEXEC	*program* is not in a recognizable executable file format, was for the wrong platform, or contained invalid fields.
ENOMEM	Insufficient virtual memory is available.
E2BIG	The total size of the argument strings is too large.
EIO	A hard I/O error occurred.
EFAULT	One of the args is an invalid pointer.

Sys_fork

- Duplicate the current process.
  - Child process will have unique process id.
- int sys_fork(struct trapframe *tf, pid_t *retval)
- Child process returns 0.
- Parent process returns the pid of the child process.
- In case of an error, do not create a new child process and return -1.
- Most of the work is already done in thread_fork. Add the following:
  - Create new process with a new pid. Add it to your process table.
  - Copy the trapframe.
  - Copy the address space.
  - Call thread_fork.

Sys_fork

- Implement md_forkentry
    - Child specific.
    - Parent’s trapframe and address space are passed as arguments.
    - Create new child trapframe.
    - Set tf_a3 to 0.
    - Get the assigned child pid from parent’s trapframe tf_v0 and assign it to the pid of the current process (since we are executing md_forkentry, this is child).
    - Set the trapframe’s tf_v0 to 0.
    - Increment tf_epc by 4.
    - Copy the passed address space to the current process address space and activate it.
    - Give the control back to the usermode.
        - Md_usermode and pass the new trapframe.

Sys_fork errors

EAGAIN	Too many processes already exist.
ENOMEM	Sufficient virtual memory for the new process was not available.

Sys_waitpid

- Wait for the process with pid to exit.
- Return its exit code with integer pointer *status*.
- Int sys_waitpid(pid_t pid, userptr_t status, int options, pid_t *ret)
    - tf->tf_a0, (userptr_t) tf->tf_a1, tf->tf_a2, &retval
- You need a mechanism for processes to show *interest* into each other.
    - You can add restrictions on which processes can show interest.
    - Make sure to prevent deadlocks by either setting restrictions to prevent it or to implement a mechanism to detect it.
- Return the pid with *status* assigned to exit status on success.
- If error, return -1 and set the ret pointer to the error code.

Sys_waitpid errors

EINVAL	The *options* argument requested invalid or unsupported options.
EFAULT	The *status* argument was an invalid pointer.

Sys_exit

- Causes the current process to terminate.
- The process id of the exiting process cannot be reused if there are other processes *interested* in it.
    - Do not put the exited pid back to available pid pool blindly.
- Void sys__exit(int code)
    - Code is the exitcode that will be assigned.

Scheduler

- Currently os161 has single queue round-robin scheduler.
- You can modify hardclock.c to have another counter that counts in HZ/2.
- Mostly scheduler.c will be edited.
  - Add a new queue.
  - Add each process a priority and modify make_runnable to match the thread and queue level according to its priority.
  - Modify the scheduler function such that the chances of picking higher level queue will increase.

Testing

- Os161/man/testbin has the details about given tests.
  - Contains html files.
  - Read them carefully and understand what needs to be implemented to pass the tests.
  - Be careful: some of them requires VM management to work.
- Forktest is very useful.
- Also test cp example in the assignment description.
- Shell implementation is given but not necessary.
  - You can call the tests by p /testbin/forktest
- A basic sys_write and sys_read also provided. These will be necessary for testing.

Testing

- Build you own tests.
- Repeat some of the tests with your new scheduler enabled.
- Make sure to include all the test outputs in your submission.

Thank you