

Section 1

syscalls involved in cowtest:

Syscall	Kernel data structures	APIs	Eventsk
fork()	process table, locks, trapframe, kernel stack, proc, page table, File table, Inode (cwd)	myproc(), allocproc(), kalloc(), uvmcopy(), freeproc(), filedup(), idup(), safestrcpy(), acquire(), release()	<ol style="list-style-type: none"> 1. Identify parent using myproc(). 2. Allocate child using allocproc() 3. Copy address space using uvmcopy(). 4. Copy trapframe to child. 5. Set chil 6. Duplicate file descriptors using filedup(). 7. Duplicate cwd using idup(). 8. Copy process name using safestrcpy(). 9. Set parent-child relationship. 10. Mark child runnable. 11. Return child PID.
sbrk()	proc, lock, page table	argint(), myproc(), growproc(), uvmmalloc(), uvmdealloc()	<ol style="list-style-type: none"> 1. Read increment n from user. 2. Save current size old_sz. 3. Call growproc(n). 4. If n > 0, call uvmmalloc(). 5. If n < 0, call uvmdealloc(). 6. Return old_sz.
exit(s)	proc, lock, file table, Inode	fileclose(), iput(), reparent(), wakeup(), sched()	<ol style="list-style-type: none"> 1. Close all open files. 2. Release cwd inode. 3. Acquire wait_lock. 4. Reparent children to initproc. 5. Set exit status and mark ZOMBIE.

			6. Wake parent. 7. Call sched().
wait(addr)	process table, lock	argaddr(), copyout(), freeproc(), sleep()	1. Acquire wait_lock. 2. Scan process table for children. 3. If zombie found, copy status and free child. 4. If none exited, call sleep(). 5. Return child PID.
pipe(fds)	pipe, file, FD table	pipealloc(), filealloc(), fdalloc(), copyout()	1. Allocate pipe buffer. 2. Allocate two file objects. 3. Configure read/write ends. 4. Attach files to pipe. 5. Allocate file descriptors. 6. Copy fds to user.
read(), write()	file, pipe, inode	argfd(), fileread(), filewrite(), piperead(), pipewrite(), copyin(), copyout()	1. Validate file descriptor. 2. Dispatch to file or pipe handler. 3. Sleep if buffer full/empty. 4. Copy data between user and kernel. 5. Update offsets or pipe pointers.
getpid()	proc	myproc()	1. Get current process. 2. Return PID.
pause(n)	ticks, tickslock	sleep(), wakeup()	1. Record current ticks. 2. Sleep while ticks < target. 3. Wake on timer interrupt.

Section 2

For sharing the parent's physical pages with the child, we need to map the child's page table entries to the parent's physical pages and set the PTE_W flag to 0 to make them non-writable. Also, a new flag, PTE_COW, is needed to identify these pages as being copy-on-write, allowing distinguishing from read-only pages. Also a physical page reference count is needed to be defined in kalloc to keep track of how many page tables share each physical page. It needs to be incremented for each shared mapping created during fork (specifically `uvmcopy()`) to account for the child process' access. Its lifetime in memory could be controlled by this count, which if it drops to 0 its freed up.

When either process attempts to write to these shared pages, it should trigger a page fault. A function to intercept this fault and invoke `vmfault()` which if the page's PTE_COW is set to 1, allocates a new physical page, copies the original data into it, and updates the process's PTE to point to this new copy with PTE_W = 1 and PTE_COW = 0 (reset) permissions. We can decrement the reference count of the old physical page, which will be freed if the ref count reaches 0.

Data Structures that need to be created/added

1. A reference count array of size (no of pages) to track how many processes point to each physical page.
2. A PTE_COW flag in the page table to identify pages handled by the COW mechanism.