

1.

Currently, the cowtest function covers 4 tests:

1. simpletest() which tests whether a fork will succeed when the parent has used 66% memory (normally it would fail since the Child won't have access to the required memory, but with shared pages in CoW fork, it should succeed)

2. threetest() has three processes created from 2 successive forks() again, and all three processes write to 80%, 50% and 100% of memory resp (for p2, p1 and p). It checks whether the writes that happened did indeed correspond to the current process's pid and not the children's. It also tests whether the pages are freed properly after exit. For this, the refcount also needs to work correctly.

3. filetest() It tests whether a new page is created on write from the kernel when it tries to write to a shared page (similar to 2).

4. forkforktest() checks for race conditions when many processes try to update a variable (the refcount for the shared page) in parallel (unless locks are used). It can read to bad values in refcount which could cause issues while freeing up the memory. The wrongly freed memory can get assigned to a new process modifying the data. Also, if it was mistakenly stuck at refcount > 0, it's never freed and cannot be used.

Some other tests that could be considered for verifying correctness of the CoW fork():

1. When the child created after a CoW fork calls exec() immediately, it creates a new address space for the child. Thus the old shared pages with the parent are no longer shared with the child and the refcounts should be decremented and its CoW state inherited from the parent should be cleaned up.
2. Similarly, when a child forked tried to free some of its memory which is shared with the parent, the memory shouldn't be freed before checking the refcount.
3. We can also check whether CoW fork manages the stack memory properly for a parent process, when it forks and the child tries to write to a variable in the shared stack, in the same way as heap memory.

2. Consider the following microbenchmarks for the CoW fork in order to assess performance

Hypothesis 1: The time to complete the CoW fork system call should be roughly constant regardless of parent address space size.

Experiment: Measure CoW fork sys call duration for different sized address spaces of processes.

CoW fork is proposed as a performance optimization. Since it doesn't make a copy of all pages right during the fork() for the child process, the time for the CoW fork time should not be dependent on the parent's address space size (unlike normal fork, where it would increase with address space size)

Hypothesis 2: The memory consumption after the CoW fork should be incremented from pre-fork, by much less than the address space size, as would be the case with normal fork.

Experiment: Measure the memory usage before and after CoW fork, it should be nearly constant regardless of the parent process address space size (i.e. repeat for different address space sizes)

Since in CoW fork, at max, we will need a few pages for storing page table entries for pointers to the parent's physical pages, the memory consumption will have increased but this increase will be constant regardless of the parent's address space size since pages aren't being copied on fork.

Hypothesis 3: With the introduction of a page fault whenever the child process attempts a write on a shared page, the time to write should be a bit more than the original, but overall time to fork should still be much lesser since this time is much lesser as compared to the time saved copying in a normal fork. (asymptotically)

Experiment: Measure the duration of a single memory write that triggers a CoW fault and compare it with the duration for a write in a normal fork. It is expected to be slightly higher, but still negligible as compared to the time saved during the fork, given the typical # of writes.

Generally, as many processes only touch a small subset of their total memory pages after forking, the time saved by not copying the untouched pages during fork will dominate the overhead of handling faults on the written pages.

A macrobenchmark that also could be tested: We test the time needed for a script that executes fork and exec multiples times in the normal and CoW fork and it should reflect the real-world benefits of using a CoW fork, as it would need much less time as compared to the normal fork, as its not actively copying pages during the fork.

