Architektury systemów komputerowych

Lista zadań nr 13

Na zajęcia 8 czerwca 2022

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

Zadanie 1. Na podstawie [1, §5.14.1] odpowiedz na następujące pytania. W jakim celu **profiluje** się programy? Jakie informacje niesie ze sobą **profil płaski** [2, 5.1] i **profil grafu wywołań** [2, 5.2]? Czemu profilowanie programu wymaga zbudowania go ze specjalną opcją kompilatora «`-pg`»? Na czym polega **zliczanie interwałowe** (ang. *interval counting*)?

Zadanie 2. Pobierz ze strony przedmiotu archiwum «`ask22_lista_13.tgz`», rozpakuj je i zapoznaj się z kodem źródłowym w pliku «`dictionary.c`». Powtórz eksperyment z [1, §5.14], tj.: uruchom program, poczekaj na wyniki *profilowania*, a następnie na podstawie wydruku wskaż kandydata do optymalizacji. Zmień dokładnie jeden z argumentów przekazywanych do polecenia «`make gprof`»: `SIZE=N`, `HASH=0/1/2`, `FIND=0/1/2`, `LOWER=0/1`, `QUICK=0/1`. Odpowiednia opcja zmieni rozmiar tablicy mieszającej `N` lub ustali wersję implementacji jednej z kluczowych funkcji w programie. Wyjaśnij co zmieniło przekazanie danej opcji. Powtarzaj powyższe kroki tak długo, aż uzyskasz najkrótszy czas wykonania programu.

UWAGA: Należy być przygotowanym do szczegółowego wyjaśnienia wydruku programu profilującego!

Zadanie 3. Dla optymalnej konfiguracji z poprzedniego zadania uruchom polecenie «`make callgrind`». Następnie wczytaj plik wynikowy «`callgrind.out`» przy pomocy nakładki graficznej «`kcachegrind`». Wyświetl **graf wywołań funkcji**, a następnie zidentyfikuj wywołania biblioteki standardowej języka `C`, w których program spędza najwięcej czasu. Wyjaśnij do czego służą te funkcje. Jak można byłoby zoptymalizować miejsca, w których korzystamy z tych funkcji?

Uwaga! W kolejnych dwóch zadaniach polecenia należy wykonywać z uprawnieniami administratora!

Zadanie 4. Posługując się poleceniem «`lspci -v`» wyświetl listę urządzeń podpiętych do *magistrali PCI*. W wydruku zidentyfikuj: regiony **fizycznej przestrzeni adresowej**, przez które procesor komunikuje się z urządzeniami wejścia-wyjścia; numery **przerwań**, które urządzenia wejścia-wyjścia zgłaszają procesorowi. Przy pomocy polecenia «`cat /proc/iomem`» wydrukuj mapę *fizycznej przestrzeni adresowej* swojego komputera. Zidentyfikuj w niej pamięć operacyjną oraz pamięć należącą do *urządzeń wejścia-wyjścia*, tj. karty graficznej, sieciowej, dźwiękowej, kontrolera dysków. W wydruku polecenia «`cat /proc/interrupts`» wskaż, który z procesorów odbiera przerwania od w/w urządzeń.

Zadanie 5. Ściągnij repozytorium `dwks/pagemap`[1]. Skompiluj program «`pagemap2`» i przy jego pomocy wyświetl zawartość **tablicy stron** procesu o numerze «`pid`» dla przedziału adresów wirtualnych, który należy wybrać na podstawie wyjścia z polecenia «`pmap`». Przetłumacz kilka **adresów wirtualnych** posługując się **numerem ramki** (ang. *page frame number*). Wskaż region pamięci w *mapie fizycznej przestrzeni adresowej*, gdzie trafił przetłumaczony adres.

Wskazówka: Zakładamy, że rozmiar strony to 4096 bajtów.

Zadanie 6. Czasami procesor nie może przeprowadzić **translacji** dla adresu wygenerowanego w trakcie wykonywania instrukcji. Wymień co najmniej cztery przypadki wykonania instrukcji «`mov`», dla których procesor zgłosi **wyjątek** *błędu dostępu do pamięci*. Które z wymienionych przypadków zawsze spowodują zakończenie programu z błędem? Za co odpowiada procedura obsługi **błędu strony** (ang. *page fault*)? Co się dzieje przed i po wykonaniu tej procedury?

[1] `https://github.com/dwks/pagemap`

Zadanie 7. Procesor `x86-64` używa **rejestru sterującego** `CR3`[2] do przeprowadzania *translacji adresów*. Co przechowuje ten rejestr? Czemu jest on dostępny wyłącznie dla programów działających w **trybie uprzywilejowanym** (ang. *supervisor mode*)? Czy moglibyśmy zagwarantować **izolację**, jeśli pamięć przechowująca tablicę stron była dostępna w **trybie użytkownika**?

Zadanie 8. Przy pomocy polecenia «`ps -e -o pid,rss,vsz,cmd`» wydrukuj listę wszystkich procesów wraz z zadeklarowanym rozmiarem **wirtualnej przestrzeni adresowej** i zbioru rezydentnego, odpowiednio `vsz` i `rss`. Użyj zdobytych danych do obliczenia rozmiaru pamięci wirtualnej używanej przez wszystkie procesy. Na podstawie wydruku polecenia «`free`» określ bieżące użycie i całkowity rozmiar pamięci fizycznej. Wyjaśnij w jaki sposób **stronicowanie na żądanie** (ang. *demand paging*) i **współdzielenie pamięci** pozwalają systemowi operacyjnemu na oszczędne zarządzanie pamięcią fizyczną.

Zadanie 9. Przyjrzymy się wydrukowi z poprzedniego zadania oraz z polecenia «`free`». Wskaż bieżące użycie i całkowity rozmiar **pamięci wymiany** (ang. *swap*). Czym różni się **zbiór roboczy** (ang. *working set*) procesu od **zbioru rezydentnego** (ang. *resident set*)? Czemu system nie podaje tego pierwszego? Kiedy system operacyjny używa wymiany do **pamięci drugorzędnej** (ang. *backing storage*)? Co dzieje się z systemem, jeśli łączny rozmiar zbiorów roboczych wszystkich procesów jest większy niż rozmiar pamięci fizycznej?

Wskazówka: Pamiętaj, że pamięć wirtualna jest programową realizacją pamięci podręcznej dla stron.

Literatura

[1] „*Computer Systems: A Programmer's Perspective*”
Randal E. Bryant, David R. O'Hallaron; Pearson; 3rd edition, 2016

[2] „*gprof: a Call Graph Execution Profiler*”
Susan L. Graham , Peter B. Kessler , Marshall K. McKusick; 1982
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.4995

[2] `https://en.wikipedia.org/wiki/Control_register#CR3`