Systemy operacyjne

Ćwiczenia 4

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Stallings: 3.1 – 3.5, 4.1, 4.2
- Tanenbaum: 2.1, 2.2
- Silberschatz: 3.1 – 3.3, 4.1, 4.3, 4.5, 4.6
- Love: 3, 4 (fragmenty)
- Maurer: 2.1 – 2.4

Polecamy również zapoznanie się z odpowiednimi treściami na OS Development Wiki.

Zadanie 1

Opisz szczegółowo **mapę pamięci** procesu w systemie Linux (`x86-32`). Zdefiniuj pojęcie **obrazu procesu**. W jakiej przestrzeni adresowej działa jądro? Czemu z reguły pierwsze kilka megabajtów **przestrzeni użytkownika** jest niedostępne dla programu? Jakie niebezpieczeństwa niesie ze sobą umieszczanie kodu i danych zawsze pod takimi samymi adresami w przestrzeni procesu? Jaki problem sprawia ładowanie **bibliotek dynamicznych** do przestrzeni adresowej procesu?

Zadanie 2

Proces to nie tylko przestrzeń adresowa i **kontekst**, ale też cały zestaw **zasobów**. Zapoznaj się z zawartością **bloku kontrolnego procesu** (**PCB**) jądra systemu Linux (`task_struct`) lub FreeBSD (`proc`). Na tej podstawie podziel na klasy rodzaje zasobów / informacji skojarzonych z procesem. Zauważ, że Linux nie definiuje osobnych struktur danych dla wątku i procesu, dlaczego? Czemu każde **zadanie** posiada osobny **stos w przestrzeni jądra**?

Zadanie 3

Opisz **siedmiostanowy model procesu**. Wyjaśnij, które przejścia w automacie opisującym ten model są rezultatem działań podejmowanych przez (a) system operacyjny (b) proces użytkownika. Podaj jakie akcje wymuszają zmianę stanów? Które z decyzji po stronie systemu operacyjnego będą podejmowane przez **planistę krótkoterminowego** i **długoterminowego**.

Zadanie 4

Czym różni się tworzenie procesów w systemie Linux i Windows NT? Rozważ wady i zalety obu rozwiązań. Wytłumacz na czym polega mechanizm **kopiowania przy zapisie**? Jakie flagi należy przekazać do wywołania systemowego **`clone()`** aby utworzyć (a) proces (b) wątek? Czy inne wariantu użycia `clone()` mogłyby mieć sens?

Zadanie 5

Na przykładzie systemu Linux opisz dokładnie proces uruchomienia programu z dysku – od wprowadzenia polecenia w powłoce `bash` po wejście do funkcji `main()`. Które z poszczególnych etapów przebiegają po stronie jądra, dynamicznego konsolidatora, a za które odpowiada uruchamiany program? Upewnij się, że nie pomijasz żadnego ważnego etapu – np.: tworzenie przestrzeni adresowej, ładowanie bibliotek dynamicznych, **procedura startowa** `crt0`.

Zadanie 6

Opisz jak przebiega **przełączanie procesów** w systemie Linux – uwzględnij stan wszystkich zasobów! Czemu na architekturze x86 (na wielu innych też) należy **opróżnić TLB**? Czym różni się **przełączanie kontekstu** od **przełączania trybu pracy**? Gdzie jądro Linuksa zachowuje rejestry przy przejściu do **przestrzeni jądra**?

Zadanie 7

W systemach uniksowych proces może zakończyć swoje działanie wywołując funkcję `exit()` lub otrzymując **sygnał**. Wskaż podobieństwa i różnice między sygnałami a **przerwaniami**. Jakie zdarzenia / sytuacje mogą spowodować wysłanie sygnału kończącego działanie procesu? Czy są takie zdarzenia, które normalnie uznaje się za błąd programu, ale mogą służyć do implementacji pewnych funkcjonalności aplikacji? Jeśli tak, to podaj przykład.

Zadanie 8

Opisz różnice między **wątkami przestrzeni jądra** (KLT), a **wątkami przestrzeni użytkownika** (ULT) – rozważ zalety i wady obu podejść. Jak **biblioteka ULT** kompensuje brak wsparcia jądra dzięki **obwolutowaniu**? Opisz model hybrydowy bazujący na **aktywacjach planisty** i pokaż, że może on połączyć zalety KLT i ULT.

Zadanie 9

Mimo swej popularności, wątki przestrzeni jądra implikują wiele interesujących niejasności. Czy proces utworzony przy pomocy `fork()` **dziedziczy** wątki? Jakie problemy to sprawia? Globalna zmienna **`errno`** przechowuje błąd ostatniej operacji – czy wątki mogą ją sobie nawzajem nadpisywać? Co to jest **TLS** (ang. *thread local storage*)? Użytkownik przerywa program z klawiatury – który wątek obsłuży **sygnał SIGINT**? Wątki w danym procesie współdzielą stertę – co jeśli wszystkie na raz próbują pobrać blok pamięci za pomocą funkcji `malloc()`?