Systemy operacyjne 2016

Lista zadań nr 3

Na zajęcia ?

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- **OS7e** – Stallings (siódme wydanie): 3.1 – 3.5, 4.1, 4.2
- **MOS4e** – Tanenbaum (czwarte wydanie): 2.1, 2.2

UWAGA! W trakcie prezentacji rozwiązań należy zdefiniować i wyjaśnić pojęcia, które zostały oznaczone **wytłuszczoną** czcionką.

Zadanie 1. Opisz **mapę pamięci** (ang. *memory layout*) procesu w systemie Linux x86–64. Zdefiniuj pojęcie **obrazu procesu**. W jaki sposób osiągnięto izolację między procesem użytkownika, a jądrem systemu? Czemu, niezależnie od platformy, pierwsze kilka megabajtów przestrzeni użytkownika jest niedostępne dla programu? Wielokrotnie ładujemy proces do pamięci – jakie niebezpieczeństwo niesie ze sobą umieszczanie kodu i danych zawsze pod tymi samymi adresami wirtualnymi?

Zadanie 2. Zapoznaj się z zawartością **bloku kontrolnego procesu** (ang. *Process Control Block*) i **wątku** (ang. *Thread Control Block*) jądra systemu FreeBSD (struktury proc[1] i thread[2]). Zasoby i informacje zawarte w `PCB` oraz `TCB` pogrupuj (jak w tabeli 3.5, OS7e), po czym wyjaśnij do czego są potrzebne. Na podstawie poczynionych obserwacji określ różnice między procesem i wątkiem.

Zadanie 3. Wskaż podobieństwa i różnice między **sygnałami**, a przerwaniami. Kiedy proces **odbiera sygnały**? Co musi zrobić proces by **wysłać sygnał** albo **obsłużyć sygnał**? Których sygnałów nie można **zignorować**? Jądro zgłasza procesowi błąd związany z wykonaniem kodu poprzez **dostarczenie** odpowiedniego sygnału, co normalnie kończy wykonanie procesu. Podaj przykład, w którym obsłużenie sygnału **SIGSEGV** lub **SIGILL** może być świadomym zabiegiem programisty.

Zadanie 4. Przedstaw automat opisujący **stan procesu** w systemie *Linux* (rysunek 4.16, OS7e). Jakie akcje lub zdarzenia wymuszają zmianę stanów? Należy właściwie rozróżnić **sygnały synchroniczne** od **asynchronicznych**. Wyjaśnij, które przejścia mogą być rezultatem działań podejmowanych przez: jądro systemu operacyjnego, kod sterowników, proces użytkownika albo administratora.

Zadanie 5. Opisz różnice między **wątkami przestrzeni jądra** (ang. *kernel-level threads*), a **wątkami przestrzeni użytkownika** (ang. *user-level threads*), a nastepnie 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 6. Wyjaśnij różnice w tworzeniu procesów w systemie *Linux* i *WindowsNT*, po czym rozważ zalety i wady obu rozwiązań. W kontekście funkcji **fork()** wytłumacz na czym polega mechanizm **kopiowania przy zapisie** (ang. *copy-on-write*)? System *Linux* implementuje `fork()` przy pomocy wywołania systemowego **clone()**. Jak zatem utworzyć proces albo wątek? Zastanów się czemu mogą służyć kombinacje pozostałych flag wywołania `clone()`.

[1] `http://fxr.watson.org/fxr/source/sys/proc.h?v=FREEBSD10#L490`
[2] `http://fxr.watson.org/fxr/source/sys/proc.h?v=FREEBSD10#L205`

Zadanie 7 (bonus). Na przykładzie systemu Linux opisz zgrubnie proces uruchomienia programu z dysku, tj. 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 kod uruchamianego programu? Upewnij się, że nie pomijasz żadnego ważnego etapu, tj. tworzenia przestrzeni adresowej, ładowania bibliotek dynamicznych, procedury startowa `crt0`.