Do labów z SO powinniście się przyłożyć, bo uczycie się na nich umiejętności rzeczywiście przydatnych w codziennym życiu.

W tym pdfie zapiszę tylko co ciekawsze instrukcje + porady do rozwiązania zadań.

Lab11:

Celem ćwiczeń jest uruchomienie Linuksa i aplikacji webej (np. w PHP) w kontenerze lub kontenerach na Windowsie.

Wariant 1. (2 punkty)

Pojedynczy kontener z prostą stroną/aplikacją (np. w PHP). Po uruchomieniu kontenera, strona powinna być dostępna porcie 8080 z poziomu Windowsa.

Wariant 2. (3 punkty)

Pojedynczy kontener z prostą stroną/aplikacją (np. w PHP), która korzysta z bazy danych. Po uruchomieniu kontenera, strona powinna być dostępna porcie 8080 z poziomu Windowsa i powinna wyświetlać dane pobrane z bazy danych.

Wariant 3. (4 punkty)

Dwa kontenery uruchamiane za pomocą Docker Compose. Jeden kontener zawiera prostą stronę/aplikację (np. w PHP), dostępną na porcie 8080 z poziomu Windowsa. Aplikacja pobiera dane z dowolnej bazy danych (MongoDB, PostsgreSQL, MySQL) uruchomionej w drugim kontenerze.

UWAGA! Wariant 2 może się okazać bardziej czasochłonny niż wariant 3.

Umieszczam go tutaj dla potomności, w praktyce mało kto je wykonał na 3 bądź 4 punkty. W niektórych grupach lab został anulowany z automatu.
Docker na SO pojawił się pierwszy raz w 2025, oczekujcie zmian i eksperymentów

Lab12:

Zadanie z szeregowania zadań

Zadanie z szeregowania zadań

Celem ćwiczenia jest praktyczne przećwiczenie zagadnień związanych z szeregowaniem zadań w jądrze systemu operacyjnego, przedstawionych w eKursie 3 (https://enauczanie.pg.edu.pl/moodle/mod/page/view.php?id=2141243).

Zadanie można wykonać albo w języku C albo w języku Python, pobierając odpowiednią wersję programu symulatora:

- C - https://github.com/AndrewidRizk/Kernel-Scheduler/tree/main
- Python - https://git.pg.edu.pl/p18685/so-scheduler

Zadanie polega na dopisaniu kolejnego algorytmu szeregowania zadań i zademonstrowanie jego działania prowadzącemu.

Wariant 1. (2 punkt)

Dodanie algorytmu Shortest Time-to-Completion First (stcf).

Wariant 2. (3 punkty)

Dodanie algorytmu Stride Scheduling (stride) lub MLFQ (3 kolejki, bez priority boost, zadania zawsze wykorzystyją cały przydzielony czas).

Najłatwiej wziąć istniejące algorytmy w tym symulatorze(korzystałem z C) a następnie zrobić z nich hybrydę. Stride jest łatwiejszy niż MLFQ, przynajmniej w mojej opinii(napisałem oba)

Lab13:

Zaawansowane operacje wejścia-wyjścia

Podstawowe operacje na plikach zwykłych są dostarczane przez wiele bibliotek, w tym przede wszystkim standardową bibliotekę języka C, poprzez nagłówek `stdio.h`. Pozwala ona na prosty odczyt czy zapis plików zwykłych, dostępnych w niemal każdym systemie operacyjnym, niezależnie czy jest to Windows czy GNU/Linux. W systemach zbudowanych na podstawie Uniksa jej funkcje takie jak `fread` czy `fopen` są wykonane z użyciem bardziej podstawowych wywołań systemowych typu `open` oraz `read`. Do wykonania wielu programów jest to w zupełności wystarczające i nie ma potrzeby korzystać bezpośrednio z wywołań. Jeśli jednak wykonywane operacje mają dotyczyć katalogów czy plików specjalnych praktycznie niezbędne jest użycie bardziej zaawansowanych narzędzi.

Przegląd zawartości katalogu

Katalogi w systemie GNU/Linux teoretycznie nie różnią się od innych plików. W praktyce jednak użycie w ich kontekście wielu podstawowych funkcji takich jak `read` kończy się niepowodzeniem. Jest to spowodowane faktem, że ten system operacyjny obsługuje bardzo wiele różnych systemów plików, w tym między innymi ext4, btrfs, f2fs czy nawet NTFS. W każdym z tych systemów struktura wpisu katalogowego może być inna, stąd istnieje potrzeba ujednolicenia interfejsu. Ta rolę pełnią na przykład funkcje `opendir` i `readdir`. Z ich pomocą można pozyskać podstawowe informacje o pliku zapisane w tzw. wpisie katalogowym, czyli min. jego nazwę czy numer i-węzła.

Odczyt statusu plików

Wpisy katalogowe w GNU/Linuksie zawierają stosunkowo niewiele informacji. Nie pozwalają chociażby zidentyfikować na którym dysku znajduje się dany plik. Jest to o tyle trudne, że GNU/Linux korzysta z jednolitego systemu plików, stąd na przykład katalogi / i /home mogą być zarówno na jednym fizycznym dysku, ale równie dobrze mogą być też na dwóch zupełnie różnych urządzeniach. Żeby to sprawdzić, istnieje funkcja `lstat`, która na postawie podanej ścieżki zwraca wspomniane informacje, tak więc instrukcja `lstat(".")` umożliwia sprawdzenie numeru dysku, na którym znajduje się aktualny katalog a `lstat("/")` analogicznego numeru dla korzenia. Niestety w zwracanej przez `lstat` strukturze brakuje nazwy pliku, dlatego jeśli chcielibyśmy sprawdzić jaką konkretnie nazwę ma bieżący katalog, musimy skorzystać z wpisów katalogowych, opisanych wyżej.

Podsumowanie

Oprócz ww. funkcji istnieje naturalnie wiele innych, jak chociażby `chdir` do zmiany bieżącego katalogu roboczego, czy liczne funkcje modyfikujące system plików `chmod`, `chown` itp. Szczegółowych informacji o nich, w tym dokładnej składni, najlepiej szukać w podręczniku systemowym, w rozdziale drugim.

Zadanie

1. Korzystając z mechanizmów wbudowanych w system plików, napisz w języku C polecenie `pwd` wyświetlające pełną ścieżkę do bieżącego katalogu. Nie używaj funkcji `getcwd` ani plików z `/proc` (2 pkt). Uwzględnij sytuację, w której bieżący katalog znajduje się na innym dysku, niż korzeń systemu plików (3 pkt).

Chyba najtrudniejsze zadanie laboratoryjne w tym semestrze tylko przez to, jak skomplikowany jest opis xD
W praktyce wywołujecie kilka razy lstat i chdir oraz zapisujecie wyniki operajcji.

Wskazówka nr 1: Poczytajcie o tym co(i jak) zwraca lstat, w szczególności o polach `st_dev` i `st_ino`
Wskazówka nr 2: Z rodzica można odczytać nazwę dziecka
Wskazówka nr 3: chdir na roocie nic nie zrobi, pozostaniecie na roocie

Lab14:

Liczniki zdarzeń

System operacyjny do zarządzania zasobami potrzebuje informacji o zużyciu zasobów. Część z nich, jak na przykład czas spędzony na wykonywaniu poszczególnych procesów, jest zbierana programowo, przez samo jądro, pozostałe, jak liczba wykonanych instrukcji, wiąże się z użyciem dedykowanych rejestrów sprzętowych. Dostęp do jednych i drugich jest możliwy poprzez narzędzie `perf`. Na przykład zużytą energię przez wszystkie procesory w ciągu jednej sekundy można odczytać poniższym poleceniem:

```
perf stat -I 1000 -a -e /power/energy-pkg/
```

Aby pobrać wartość konkretnego licznika, trzeba znać jego nazwę oraz posiadać odpowiednie uprawnienia. Polecenie `perf list` pozwala wyświetlić wszystkie dostępne liczniki w danym systemie. Uprawnieniami natomiast zarządza się przez `sysctl`. Jeśli w pliku `/etc/sysctl.conf` znajduje się linia zawierająca `kernel.perf_event_paranoid=0` oznacza to, że możemy czytać z praktycznie wszystkich liczników będących w systemie.

Oprócz wspominanego wyżej gotowego narzędzia w powłoce, istnieje także API w języku C. Jest ono dokładnie opisane w podręczniku systemowym, na stronie o nazwie `perf_event_open`. Każde zdarzenie posiada szereg atrybutów zgromadzonych w strukturę `perf_event_attr`. Dwa z nich, `type` i `config` są szczególnie istotne. Drugi wskazuje na numer zdarzenia, natomiast pierwszy podaje numer jednostki monitorującej. W GNU/Linuksie może istnieć wiele różnych źródeł zdarzeń, stąd konieczne jest wskazanie konkretnego. Dzięki temu ten sam numer zdarzenia w obrębie różnych jednostek może odnosić się do zupełnie czegoś innego. Podstawowe numery są zdefiniowane jako stałe w plikach nagłówkowych, jednak nie jest to ich pełnia lista. GNU/Linux jest systemem modułowym, co oznacza między innymi możliwość ładowania sterowników w trakcie pracy systemu, stąd istnieje potrzeba takiego określania numerów zdarzeń, aby było możliwe ich pobieranie już po złożeniu programu ze źródeł, w trakcie pracy. Realizuje się to poprzez specjalne pliki, zawarte wewnątrz katalogu `/sysfs`.

Dynamiczne PMU

Każdy jednostka monitorująca wydajność posiada dedykowany katalog wewnątrz `/sys/bus/event_source/devices`. Pliki opisujące PMU związane z pomiarami energii znajduje się w katalogu `/sys/bus/event_source/devices/power`. Jej numer jest zapisany w `/sys/bus/event_source/devices/power/type` natomiast w podkatalogu `events` są dane poszczególnych liczników zdarzeń. Plik `/sys/bus/event_source/devices/power/events/energy-pkg` przechowuje numer licznika o nazwie `energy-pkg`. Jeśli do poprawnego odczytu licznika są potrzebne dodatkowe informacje, są one dostępne w osobnych plikach. Na przykład `/sys/bus/event_source/devices/power/events/energy-pkg.scale` zawiera współczynnik skali przez który należy przemnożyć odczyt z licznika, aby uzyskać wartość w jednostce, która jest podana w `/sys/bus/event_source/devices/power/events/energy-pkg.unit`.

Zadanie

1. Wykorzystując wywołanie systemowe `perf_event_open`, napisz program w języku C, który poda w watach moc zużytą przez CPU (2 pkt). Program powinien wypisywać wynik co sekundę i kończyć się po naciśnięciu `Ctrl+C` wyświetlając sumaryczną energię zużytą w trakcie swojego działania (3 pkt).

Po prostu poszukajcie jak wygląda wykorzystanie perf_event_open. Składania tego wywołania jest, hm, ciekawa - najtrudniejsza część zadania to właśnie jej poprawne użycie xD

Wskazówka 1: Dokumentacja
Wskazówka 2: Przydatny blog