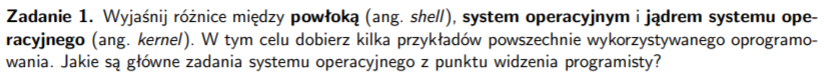
Systemy Komputerowe

**Lista A**



Powłoka (shell) – Jest to najniższy poziom interfejsu dostępny dla użytkownika. Służy np. uruchamianiu programów, które potem obsługuje system operacyjny. Powłoka to to w czym porusza się użytkownik.

System operacyjny – Jest to taki „most” pomiędzy sprzętem a oprogramowaniem. Dzięki niemu programista nie musi od podstaw pisać wszystkich operacji niskopoziomowych (np. obsługa plików na poziomie SATA), aby program mógł ich używać. Pomaga skupić się na tworzeniu faktycznego programu i udostępnia gotowe rozwiązania dla programów.

Jądro system operacyjnego – Centrum systemu operacyjnego. Wiele systemów operacyjnych może mieć to samo lub podobne jądro (np. Android i Ubuntu), chociaż mogą wyglądać inaczej i mieć inne zastosowania. Jądro odpowiada za zarządzanie hardware, pamięcią itd.

Główne zadania systemu operacyjnego:

Tworzy abstrację, dzięki której programista nie musi martwić się o niskopoziomowe operacje np. zapisywanie plików na nośnikach, przełączanie procesów

Zarządza procesami uruchomionymi na danej maszynie, przydziela zasoby odpowiednim procesom

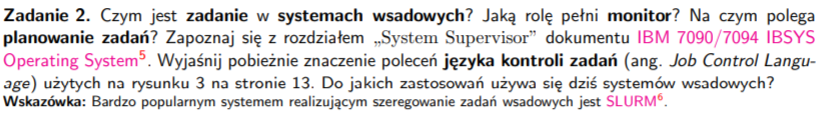
Ułatwia i skraca czas potrzebny do napisania programu (skupiamy się tylko na samym programie, nie na wszystkich funkcjach maszyny, z których korzysta)

Przykłady programów:

Klient poczty (użytkownik przegląda maile i nimi zarządza z poziomu powłoki lub GUI, klient poczty pobiera maile, wysyła je)

Aplikacja do zmiany hasła na koncie w systemie (użytkownik chce tylko zmienić hasło, do programu należy odpowiednia współpraca z SO, aby hasło zostało zmienione)

Eksplorator plików np. na Ubuntu (użytkownik może w graficznej formie zarządzać plikami, program współpracuje z SO, aby zmieniać odpowiednie bloki pamięci)



Zadanie (job) jest to program albo zbiór programów, które programista pisze w jakimś języku, wybija potem na punchcardach i takie karty przekazuje do centrum obliczeniowego w celu zrealizowania kodu.

System wsadowy pobiera wsad z wejścia (mogło to być więcej niż jedno zadanie), Monitor śledzi wykonywanie danego zadania na maszynie. System wsadowy na stale rezyduje w pamięci operacyjnej. Najczęściej po wykonaniu danego zadania przekazuje wyniki do kolejnego zadania, dla którego są to dane wejściowe.

Monitor – program, który na stałe rezyduje w pamięci i służy śledzeniu przebiegu realizacji zadania.

Planowanie zadań – Planujemy jak zadania zostaną uszeregowane w maszynie, aby wykorzystać jak najwydajniej jej moc obliczeniową.

Język kontroli zadań – Sterowany kartami o odpowiednich sygnaturach, pozwalał na wywołanie określonego zachowania maszyny.

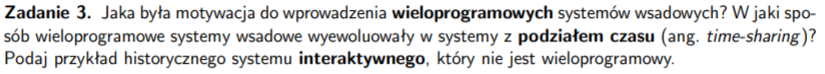
Podstawowe karty:

**$JOB** – Oznacza początek danego zadania. Mówi także systemowi jak przetworzyć dane zadanie. Ponad to, jeśli są skolejkowane zadania na wejściu $JOB oznacza także koniec poprzedniego zadania.

**$IBSYS** – Wywołanie System Supervisor w celu obsługi sekwencji kart.

**$EXECUTE** – Inicjuje uruchomienie monitora w celu wykonania zadania. Monitor potem czyta dane i uruchamia odpowiednie funkcje.

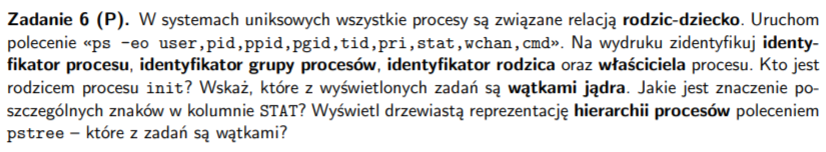
**$STOP** – Zatrzymanie wejścia i wyjścia oraz wylistowanie błędów i logów wejścia/wyjścia.

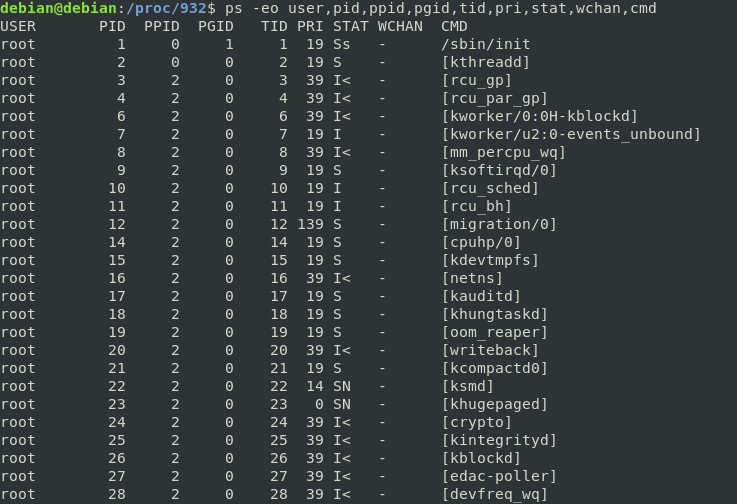


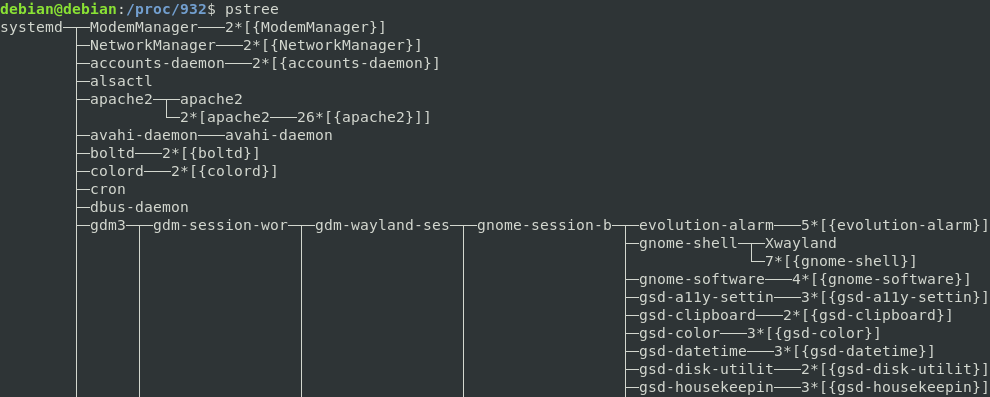
Przed wprowadzeniem wielkoprogramowych systemów wsadowych traciliśmy większą część czasu procesora na oczekiwanie na input/output. Podczas oczekiwania procesor nie robił nic, ponieważ nie miał danych do wykonania zadania. Było to duże marnotrawstwo czasu procesora. Dzięki wprowadzeniu wielkoprogramowych systemów wsadowych mogliśmy podczas oczekiwania na output dla Task1 rozpocząć wykonywanie Task2 i wrócić do Task1 kiedy już będziemy mieli przygotowane dane od przetworzenia.

Początkowo wielkoprogramowość była jednak problematyczna. Wcześniej programiści mieli maszynę całą dla siebie od momentu podania wejścia do otrzymania wyjścia. W multiprogramowości kilka zadań mogło korzystać z maszyny, przez co zadania maszyna nie musiała produkować wyników w kolejności, więc w najgorszym wypadku na swoje wyniki trzeba było czekać długo. Było to tym bardziej uciążliwe, że debugging programów trwał długo i był mozolny. Powstała idea time-sharing. Każdy użytkownik miał swój terminal. Jeśli przykładowo 20 użytkowników było podłączonych, jednak 17 w danym momencie nie korzystało z procesora, zasoby były przydzielane tak, aby obsłużyć te 3 zadania, które potrzebowały mocy obliczeniowej.

System: OS/360?







Rodzic-Dziecko: Każdy proces posiada rodzica, który go utworzył przy pomocy fork(). Wyjątkiem jest init, którego rodzicem jest kernel.

Identyfikator procesu (PID): unikalna w systemie operacyjnym liczba, identyfikująca jednoznacznie dany proces.

Grupa procesów (PGID): Zbiór procesów. Jeśli sygnał zostanie wysłany do jednego z nich, wszystkie otrzymają ten sygnał.

Identyfikator rodzica (PPID) – Identyfikator procesu, który uruchomił dany proces (rodzic)

Identyfikator właściciela (USER) – Użytkownik, który uruchomił dany proces

Rodzicem procesu init jest kernel.

Wątki jądra – Wątki, które wykonują kod jądra. Nie są powiązane z procesami użytkownika. Charakteryzują się nazwami w nawiasach kwadratowych (nie ma podane jaka jest nazwa tego co je wywołało).

Hierarchia procesów: wyświetla drzewko zależności rodzic-dziecko procesów

Znaczenie znaków w STAT:

D - uninterruptible sleep

S - interruptible sleep

R - runnable

X - dead

T - stopped

Z - zombie

I - idle kernel thread

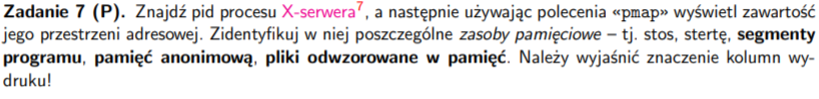
< - high priority

+ - w grupie procesów pierwszoplanowych

l - wielowątkowy

s - session loader (pierwszy proces w sesji)

Drzewiasta reprezentacja hierarchii procesów (pstree). Które to wątki? Wątki to te, które mają nazwy w nawiasach klamrowych.



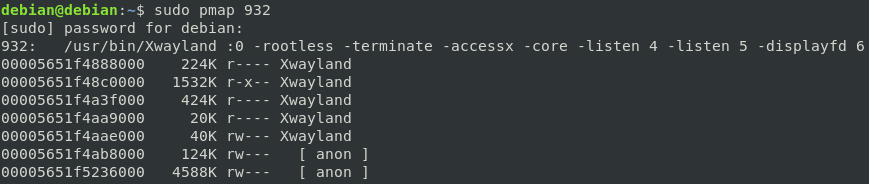
Znalezienie procesu: Ps ax | grep X

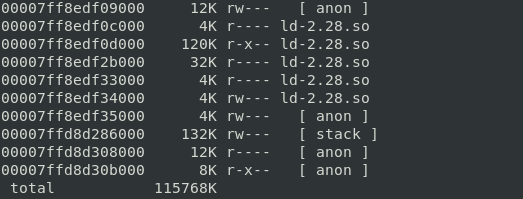


/usr/bin/Xwayland

PID: 932

Sudo pmap 932 (chociaż lepiej to widać: cat /proc/932/maps





Segmenty programu – Spójne bloki pamięci o jednolitym przeznaczeniu i atrybutach z punktu widzenia procesu ładowania i uruchamiania programu.

Segment kodu – Obszar w pamięci zawierający kod maszynowy programu.

Pamięć anonimowa – Pamięć, która nie jest powiązana z żadnym konkretnym obiektem bądź plikiem. Przykładem może być np. sterta.

Pliki odwzorowane na pamięć – Miejsce pamięci wirtualnej, które jest zmapowane z jakimś plikiem lub zasobem. Możemy wtedy traktować plik jako ciągłą pamięć, do której możemy pisać i którą możemy czytać bezpośrednio. Zmiany są zapisywane na dysku.

Stos – [stack]

Sterta – [heap]

Vdso – (virtual dynamic shared object) biblioteka współdzielona udostępniana przez jądro. Służy przyspieszeniu wykonywania syscalli.

Vvar – Read-only – Zmienne jądra

Kolumny wydruku:

Address – adres początku opisanego miejsca

Size-KBytes – Rozmiar

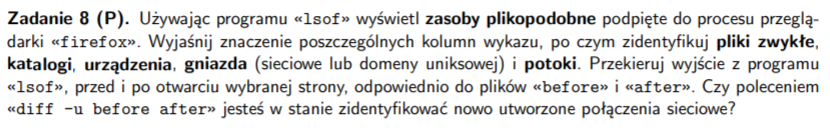
RSS – fizyczny rozmiar

Dirty – Strony „dirty” w KB

Mode – Uprawnienia

Device – nazwa urządzenia

Mapping – to na co jest zmapowana pamięć



Zasób plikopodobny (deskryptor pliku) – zasób mający deskryptor, ale nie będący prawdziwym plikiem (np. stdin stdout stderr)

Ps -ax | grep firefox

Lsof -p [pid]

|  |
| --- |
| znaczenie kolumn wykazu: |
|  |  |
|  | COMMAND - polecenie, które uruchomiło proces |
|  | PID - identyfikator procesu |
|  | USER - właściciel procesu |
|  | FD - numer deskryptora pliku lub ktoreś z poniższych |
|  | cwd - current working directory |
|  | L[nn] - library reference |
|  | err - FD information error |
|  | jld - jail directory |
|  | ltx - shared library text |
|  | M[xx] - hex memory-mapped type number |
|  | m86 - DOS Merge mapped file |
|  | mem - memory-mapped file |
|  | mmap - memory-mapped-defive |
|  | pd - parent directory |
|  | rtd - root directory |
|  | tr - kernel trace file |
|  | txt - program text (code and data) |
|  | v86 - VP/ix mapped file |
|  | Po FD pojawia się trym w jakim plik został otworzony |
|  | w - write |
|  | r - read |
|  | u - write and read |
|  | spacja jeżeli tryb jest nieznany (brak znaku |
|  | oznaczającego rodzaj blokady) |
|  | '-' jeżeli tryb jest nieznany |
|  | TYPE - typ węzła powiązanego z plikiem |
|  | DIR - katalogi |
|  | DEL - usunięty plik zmapowany |
|  | REG - plik zwykły |
|  | IPv4 - socket IPv4 |
|  | ax25 - socket Linux AX.25 |
|  | inet - Internet domain socket |
|  | sock - socket of unknown domain |
|  | unix - UNIX domain socket - służy do wymiany |
|  | danych pomiędzy procesami |
|  | x.25 - HP-UX x.25 socket |
|  | STSO - stream socket |
|  | wysyłanie i odbieranie danych. |
|  | FIFO - specjalny plik (nazwany potok), który jest częścią |
|  | systemu plików |
|  |  |
|  | plik zwykły - zwykły plik, po użyciu ls -l ma '-' w polu |
|  | trybu (np. -rw-r--r--). W lsof oznaczony przez REG. |
|  |  |
|  | katalog (directory) - struktura organizacji danych (specjalny |
|  | rodzaj pliku), która zawiera referencje do plików lub innych |
|  | katalogów. W lsof oznaczony przez DIR. |
|  |  |
|  | plik urządzenia (device file) - specjalny plik służący do komunikacji |
|  | programu z urządzeniem poprzez używanie jego sterowników |
|  | przy użyciu zwykłych syscalli wejścia/wyjścia. |
|  |  |
|  | gniazdo (socket) - dwukierunkowy punkt końcowy połączenia. Umożliwia |
|  | przesyłanie danych w obie strony. |
|  | Gniazdo posiada trzy główne właściwości: |
|  | - typ gniazda identyfikujący protokół wymiany danych |
|  | - lokalny adres |
|  | - opcjonalny lokalny numer portu identyfikujący proces |
|  | W lsof oznaczony przez sock/IPv4/unix itp. |
|  |  |
|  | potok (pipe) - mechanizm komunikacji międzyprocesowej umożliwiający |
|  | wymianę danych pomiędzy dwoma procesami. Odbywa się to najczęściej |
|  | poprzez połączenie stdio jednego procesu ze stdin drugiego. |
|  | Rozwinięciem idei potoku jest potok nazwany. Polega ono na stworzeniu |
|  | specjalnego pliku, który ma służyć jako łącznik między procesami. |
|  | W lsof oznaczony jako FIFO. |
|  |  |
|  |  |
|  | Jak zidentyfikować nowo utworzone połączenia sieciowe? |
|  |  |
|  | ps -ax | grep "firefox" |
|  | lsof -p [pid] > before |
|  | lsof -p [pid] > after |
|  | diff -u before after | grep IPv4 |