**Definicje podstawowe i obowiązkowe:  
- system operacyjny** - System operacyjny jest to zbiór programów i procedur spełniających dwie podstawowe funkcje:

zarządzanie zasobami systemu komputerowego,

tworzenie maszyny wirtualnej.

- **zasób systemu** - Zasobem systemu jest każdy jego element sprzętowy lub programowy, który może być przydzielony danemu procesowi. Zasobami system zarządza w czasie i przestrzeni.

Przez zasoby sprzętowe rozumiemy:

czas procesora,

pamięć operacyjną,

urządzenia zewnętrzne,

inne komputery powiązane poprzez sieć teleinformatyczną.

Przez zasoby programowe rozumiemy:

pliki,

bufory,

semafory,

tablice systemowe.

**- proces -** Procesem nazywamy wykonujący się program wraz z jego środowiskiem obliczeniowym. Proces stanowi podstawowy obiekt dynamiczny w systemie operacyjnym. **- powłoka (interpreter poleceń) -** Powłoka, interpreter poleceń ang. shell jest programem uruchamianym standardowo po otwarciu sesji użytkownika przez proces **login**. Powłoka jest aktywna aż do wystąpienia znaku <EOT>, który powoduje jej zatrzymanie i zgłoszenie tego faktu do jądra systemu. Każdy użytkownik otrzymuje własny i odrębny egzemplarz sh. Program sh wypisuje **monit $** na ekranie, dając znać o swojej gotowości do przyjęcia polecenia (komendy).  
**- sekcja krytyczna, -** fragment programu, w którym występują instrukcje dostępu do zasobów dzielonych. Instrukcje tworzące sekcje krytyczne muszą być poprzedzone i zakończone operacjami realizującymi wzajemne wykluczanie. **- semafor** - zmienna nazwana semaforem, inicjowana nieujemną wartością całkowitą i zdefiniowana poprzez definicje niepodzielnych operacji P(s) i V(s)

**Wstęp**  
**- tryby przetwarzania systemu komputerowego**

**tryb wsadowy**, pośredni (ang. off-line, batch), autonomiczne wykorzystanie komputera bez konieczności obecności użytkownika

+ duża przepustowość systemu komputerowego,

– możliwy długi okres oczekiwania na wyniki, ograniczone możliwości szeregowania, niemozność bieżącej kontroli procesu wykonania.

**tryb interaktywny**, bezpośredni (ang. on-line, interactive), konwersacyjne współdziałanie użytkownika z systemem komputerowym z wykorzystaniem terminala komputera.

+ szybka reakcja systemu, możliwość kontroli przebiegu procesu wykonania,

– mniejsze wykorzystanie zasobów systemu komputerowego

**tryb czasu rzeczywistego**, system, którego użytkownikiem jest proces technologiczny narzucający pewne wymagania czasowe. Dwa podejścia:

– system jest zobowiązany do reagowania na zdarzenia zewnętrzne w ustalonym nieprzekraczalnym okresie.

– system bada okresowo stan procesu technologicznego

**- rola przerwania w systemie komputerowym** mechanizm pozwalający na przerwanie bieżących działań procesora w celu obsługi pilnych zadań, takich jak reakcja na sygnały od urządzeń zewnętrznych lub zarządzanie wielozadaniowością w systemie operacyjnym. Umożliwiają efektywne wykorzystanie procesora i reagowanie na zdarzenia w czasie rzeczywistym

**- pojęcie blokady w systemie operacyjnym -** mechanizm synchronizacji, który zapobiega jednoczesnemu dostępowi wielu procesów lub wątków do tego samego zasobu lub sekcji kodu, który wymaga ekskluzywnego dostępu

**Obraz zawierający tekst, zrzut ekranu, diagram, Czcionka

Opis wygenerowany automatycznie  
- przeznaczenie montowania systemu plików -** to proces, w którym system plików zostaje udostępniony i dostępny dla użytkownika i systemu operacyjnego. Montowanie pozwala systemowi operacyjnemu na dostęp do plików i katalogów na urządzeniu przechowywania (takim jak dysk twardy, dysk SSD, pamięć USB) w sposób zorganizowany i kontrolowany. Oto główne cele i znaczenie montowania systemu plików:

Obraz zawierający tekst, diagram, zrzut ekranu

Opis wygenerowany automatycznie

**Programowanie w języku powłoki:**  
**- użytkownicy w systemie Unix -** Użytkownicy w systemie Unix

• superużytkownik (ang. superuser, root),

• pozostali użytkownicy. **- prawa dostępu do plików**

Prawa dostępu do plików w systemach operacyjnych typu UNIX określają, kto i w jaki sposób może interagować z plikiem lub katalogiem. Są one podzielone na trzy główne kategorie:

* Właściciel (owner): Użytkownik, który ma kontrolę nad plikiem.
* Grupa (group): Użytkownicy należący do tej samej grupy, co plik.
* Inni (others): Wszyscy pozostali użytkownicy.

Prawa dostępu dla każdej z tych kategorii obejmują:

* Czytanie (r):
* Zapis (w):
* Wykonanie (x):

**- znaczenie bitów SUID, SGID**

Gdy bit SUID jest ustawiony na pliku wykonywalnym, proces uruchamiający ten plik otrzymuje uprawnienia właściciela pliku (zwykle roota) na czas jego wykonania.

SGID (Set Group ID): Podobnie, gdy bit SGID jest ustawiony na pliku wykonywalnym, proces uruchamiający ten plik otrzymuje uprawnienia grupy pliku.

**- główne zmienne powłoki**

* HOME: Ścieżka do katalogu domowego użytkownika.
* PATH: Lista katalogów, w których powłoka szuka poleceń.
* PS1: Wygląd promptu powłoki.
* USER: Nazwa zalogowanego użytkownika.
* PWD: Aktualny katalog roboczy.
* SHELL: Ścieżka do aktualnie używanej powłoki.

**- parametry powłoki -** Parametry powłoki

$0 nazwa wywołanej komendy (cmd) 3 Programowanie w języku powłoki sh

$1 pierwszy argument (parametr) wywołania

$2 drugi argument (parametr) wywołania

$9 dziewiąty argument (parametr) wywołania

$\* argumenty jako jeden łańcuch znaków "$\*" = "$1 $2 .."

$@ argumenty jako osbne łańcuchy znaków "$@" = "$1" "$2" ..

$# liczba argumentów przekazanych przy wywołaniu lub przez set,

$? stan końcowy (ang. exit status) ostatnio wykonywanej komendy,

$$ numer procesu aktualnie wykonywanej powłoki,

$! numer procesu ostatnio wykonywanego procesu w tle.

$0-9 także: opcje przypisane powłoce przy wywołaniu lub przez set **- polecenia zewnętrzne a wbudowane – uzasadnienie rozróżnienia**

Postać polecenia

$ nazwa\_polecenia arg1 ... arg9

$ echo $PATH

Polecenia wbudowane

$ PATH=$PATH:/usr/local/bin

$ export PATH

• bezparametrowe set wyświetla wartości wszystkich zmiennych środowiska

• bezparametrowe export wyświetla wartości wszystkich eksportowanych zmiennych środowiska **- odczyt pliku /etc/passwd z wykorzystaniem read/set/IFS,**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

**- obsługa argumentów wywołania skryptu i funkcji**

Argumenty Skryptu: Dostępne poprzez $1, $2, ..., gdzie $1 to pierwszy argument, $2 to drugi itd. $# zawiera liczbę argumentów.

Argumenty Funkcji: Podobnie jak w skryptach, dostępne przez $1, $2, ..., w kontekście danej funkcji.

**Procesy i wątki:**  
- **pojęcia procesów współbieżnych/równoległych/rozproszonych**

Procesy Współbieżne:

Procesy współbieżne to procesy, które są wykonywane w tym samym czasie, ale niekoniecznie równocześnie. W systemach jednoprocesorowych, współbieżność jest symulowana przez szybkie przełączanie się między procesami, co daje wrażenie, że działają one równocześnie. W wieloprocesorowych, procesy mogą być faktycznie przetwarzane równocześnie, ale nadal uważane są za współbieżne ze względu na współdzielenie zasobów systemowych.

Procesy Równoległe:

Procesy równoległe to procesy, które faktycznie są wykonywane jednocześnie, zwłaszcza na systemach z wieloma procesorami (np. wielordzeniowe procesory). Równoległość odnosi się do jednoczesnego przetwarzania wielu procesów, co jest możliwe dzięki fizycznemu podziałowi pracy między różne jednostki obliczeniowe.

Procesy Rozproszone:

Procesy rozproszone to procesy, które są wykonywane na różnych, niezależnych komputerach połączonych siecią. Każdy komputer wykonuje część zadania, a komunikacja między nimi odbywa się przez sieć. Ten model jest używany w systemach rozproszonych, gdzie obciążenie jest rozdzielane na wiele maszyn, co pozwala na efektywniejsze przetwarzanie dużych zbiorów danych lub zadań wymagających dużych mocy obliczeniowych.

**- graf przejść stanów procesów w systemie Unix - prosty i złożony  
(System V),**

**Obraz zawierający tekst, Czcionka, krąg, design

Opis wygenerowany automatycznie**

**Obraz zawierający diagram, krąg, linia

Opis wygenerowany automatycznie  
- przeznaczenie i budowa wektora przerwań -** wektor przerwania zawiera adres procedury dostarczonej przez system operacyjny, z każdym urządzeniem I/O skojarzona jest lokalizacja w pamięci z informacją, skąd ma nastąpić kontynuacja wykonywania podprogramu w przypadku

wystąpienia przerwania **- porównanie własności procesów i wątków**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie  
- porównanie wątków poziomu jądra i wątków poziomu użytkownika,**

**Obraz zawierający diagram, szkic, rysowanie, linia

Opis wygenerowany automatycznie**

**- architektura wielowątkowa w systemie Solaris -** Solaris wykorzystuje cztery rozłączne koncepcje:

• Procesy - standardowe procesy systemu Unix,

• Wątki poziomu użytkownika - zaimplementowane bibliotecznie, nierozróżnialne

z punktu widzenia jądra, stanowią interfejs do współbieżności,

• Procesy lekkie (ang. Lightweight processes, LWP), LWP stanowi formę odwzorowania między wątkami poziomu użytkownika a wątkami jądra.

– każdy LWP obsługuje jeden bądź więcej wątków poziomu użytkownika odwzorowując w jeden wątek jądra,

– LWP rozróżniane i szeregowane przez jądro,

– LWP mogą być uruchomione równolegle w architekturze wieloprocesorowej,

• Wątki jądra podstawowe elementy szeregowane i rozmieszczane na procesorach.

**- metody konstrukcji serwerów** usług

Metody organizacji serwerów usług:

• serwery wielowątkowe - współbieżność, blokujące wywołania systemowe,

• procesy jednowątkowe - brak współbieżności, blokujące wywołania systemowe,

• automaty skończone - współbieżność, nieblokujące wywołania systemowe,

przerwania.

**- szeregowanie z wywłaszczaniem i bez wywłaszczania,**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie  
- szeregowanie procesów - wskaźniki jakości szeregowania dla różnych trybów  
przetwarzania,**

Wszystkie systemy

• sprawiedliwość - każdemu równą część czasu CPU,

• zgodność z polityką - praca wedle założeń,

• wyrównywanie - zbliżona zajętość wszystkich części systemu.

Systemy wsadowe

• przepustowość - maksymalizacja liczby zadań w czasie,

• czas w systemie - min. czasu między uruchomieniem a zakończeniem,

• wykorzystanie procesora - minimalizacja przerw w pracy procesora.

Systemy interaktywne

• czas odpowiedzi - możliwie szybka odpowiedź na żądanie,

• proporcjonalność - spełnianie oczekiwań użytkownika.

Systemy czasu rzeczywistego

• spełnianie wymagań - spełnianie ograniczeń czasowych,

• przewidywalność - np. unikanie spadku jakości w przekazie multimediów

**Wzajemne wykluczanie i synchronizacja:  
- wyścig i warunki wyścigu -** nazywamy sytuację, w której dwa lub więcej procesów wykonuje operację na zasobach dzielonych, a ostateczny wynik tej operacji jest zależny od momentu jej realizacji. **- warunki konieczne implementacji sekcji krytycznej -** Dla prawidłowej implementacji sekcji krytycznych muszą być spełnione następujące 3 warunki, przy czym nie czynimy żadnych założeń dotyczących szybkości działania procesów, czy też liczby procesorów:

1. wewnątrz SK może przebywać tylko jeden proces,

2. jakikolwiek proces znajdujący się poza SK, nie może zablokować innego

procesu pragnącego wejść do SK,

3. każdy proces oczekujący na wejście do SK powinień otrzymać prawo dostępu w rozsądnym czasie.

**- mechanizmy realizacji wzajemnego wykluczania z aktywnym oczekiwaniem –  
opis i porównanie**

1. Blokowanie Przerwań (Disabling Interrupts)

Działanie: Proces wchodzący do sekcji krytycznej (SK) blokuje przerwania, a wychodząc odblokowuje.

Zalety: Zapobiega interwencji innych procesów podczas aktualizacji zasobów dzielonych.

Wady: Może prowadzić do upadku systemu, jeśli przerwania nie zostaną odblokowane; nieskuteczny w systemach wieloprocesorowych.

2. Zmienne Blokujące (Lock Variables)

Działanie: Użycie zmiennej lock, która wskazuje, czy SK jest zajęta.

Problem: Problem wyścigu, ponieważ sprawdzanie i ustawianie zmiennej lock nie jest atomowe.

3. Ścisłe Następstwo (Strict Alternation)

Działanie: Procesy na zmianę wchodzą do SK, kontrolowane przez zmienną turn.

Problem: Zagłodzenie – proces może zostać zablokowany nieskończenie długo, jeśli drugi proces nie chce korzystać ze SK.

4. Algorytm Petersona

Działanie: Kombinacja ścisłego następstwa i zmiennych blokujących; każdy proces przed wejściem do SK wywołuje enter\_region z własnym numerem.

Zalety: Skutecznie rozwiązuje problem wzajemnego wykluczania bez użycia przerwań.

Wady: Nadal wymaga aktywnego oczekiwania, co może być marnotrawstwem zasobów.

5. Instrukcja TSL (Test and Set Lock)

Działanie: Sprzętowa instrukcja, która atomowo czyta i ustawia wartość blokady.

Zalety: Atomowość działania zapobiega problemom wyścigu.

Wady: Podobnie jak inne, wiąże się ze stratą czasu procesora na aktywne oczekiwanie.

Porównanie

Efektywność: Algorytm Petersona i instrukcja TSL są bardziej efektywne niż blokowanie przerwań czy ścisłe następstwo, ponieważ lepiej radzą sobie z problemami wyścigu i zagłodzenia.

Zastosowanie: Blokowanie przerwań jest ograniczone do środowiska jądra, podczas gdy inne techniki mogą być stosowane w przestrzeni użytkownika.

Koszt zasobów: Wszystkie te techniki, oprócz instrukcji TSL, wiążą się z aktywnym oczekiwaniem, co może prowadzić do nieefektywnego wykorzystania procesora.

Złożoność: Algorytm Petersona jest prostszy w implementacji niż ścisłe następstwo, ale bardziej skomplikowany niż proste zmienne blokujące.

**- mechanizmy realizacji wzajemnego wykluczania z wstrzymywaniem procesu - opis i porównanie**

1. Sleep and Wakeup

Działanie: sleep() zawiesza proces do momentu obudzenia przez inny proces za pomocą wakeup().

Zastosowanie: Przykładowo w problemie producent-konsument.

Wady: Możliwość utraty sygnału wakeup, co prowadzi do blokady.

2. Semafory

Działanie: Zmienna całkowita zliczająca sygnały wakeup. Operacje P (wait) i V (signal) zmieniają wartość semafora w sposób niepodzielny.

Zastosowanie: Oszczędniejsze od sleep/wakeup, często używane w systemach operacyjnych.

Wady: Potrzeba zarządzania kolejkami procesów czekających.

3. Monitory

Działanie: Zbiór procedur, zmiennych i struktur danych w specjalnym module. Tylko jeden proces może przebywać w monitorze.

Zastosowanie: Uproszczenie programowania współbieżnego, automatyzacja mechanizmu wzajemnego wykluczania.

Wady: Ograniczona dostępność w językach programowania, trudność implementacji w środowiskach rozproszonych.

4. Komunikaty (Message Passing)

Działanie: Wymiana informacji między procesami przez wysyłanie i odbieranie wiadomości.

Zastosowanie: Skuteczne w systemach rozproszonych, gdzie procesy nie mają wspólnej pamięci.

Wady: Złożoność zarządzania buforami wiadomości, potencjalne opóźnienia w komunikacji.

Porównanie

Efektywność: Semafory i monitory są zwykle bardziej efektywne niż sleep/wakeup, ponieważ lepiej zarządzają stanami oczekiwania procesów.

Złożoność: Monitory i komunikaty są bardziej skomplikowane w implementacji niż semafory czy sleep/wakeup, ale oferują większą kontrolę i bezpieczeństwo.

Uniwersalność: Komunikaty są najlepsze dla systemów rozproszonych, podczas gdy semafory i monitory są lepsze w środowiskach z wspólną pamięcią.

Ryzyko blokad: Sleep/wakeup ma ryzyko utraty sygnałów, podczas gdy semafory i monitory mają wbudowane mechanizmy zapobiegające takim sytuacjom.

**- producent-konsument - poprawna realizacja z synchronizacją semaforami**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, algebra

Opis wygenerowany automatycznie  
- producent-konsument - poprawna realizacja z synchronizacją monitorami,**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, algebra

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu, paragon, Czcionka

Opis wygenerowany automatycznie**

**Przykładowe zadania projektowe:**  
- proste skrypty sh (bez rozszerzeń bash):  
- skrypt, który wysyła na wyjście linia po linii tekst ze  
standardowego wejścia, poprzedzając każdą linię numerem linii,

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie  
- skrypt, który przyjmuje jako argument id użytkownika i  
wypisuje na stdin nazwę użytkownika (korzysta z relacji w /etc/passwd),

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie  
- skrypt wypisujący zadane argumenty w odwrotnej kolejności,

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie  
- opisz rezultat wykonania poleceń i grup poleceń:

**x >/dev/null 2>&1** - zarówno standardowe wyjście, jak i standardowe wyjście błędów polecenia x są ignorowane. Nie zobaczysz żadnego wyniku ani komunikatów o błędach od x.  
**x 2>&1 1>/dev/null -** Efektem tego jest, że standardowe wyjście błędów (stderr) zostanie wyświetlone, a standardowe wyjście (stdout) zostanie zignorowane. **x 2>/dev/null 1>/dev/null -** zarówno standardowe wyjście, jak i wyjście błędów są ignorowane.  
**x < y > z -** x przetwarza dane z pliku y i zapisuje wynik do pliku z.  
**x & y ; z -** x działa w tle, podczas gdy y i następnie z są wykonywane kolejno.  
**x || y -** jeśli polecenie x zakończy się niepowodzeniem (zwróci kod wyjścia różny od zero), zostanie wykonane polecenie y  
- opisz precyzyjnie działanie poniższych komend, ile i dokładnie kiedy tworzonych jest procesów?  
x > y

Uruchamia proces dla polecenia x.

Standardowe wyjście polecenia x jest przekierowywane do pliku y.

Tworzony jest 1 proces.  
x | y

Uruchamia procesy dla poleceń x i y.

Standardowe wyjście x jest przekierowywane do standardowego wejścia polecenia y.

Tworzone są 2 procesy jednocześnie.  
x < y

Uruchamia proces dla polecenia x.

Standardowe wejście polecenia x jest przekierowywane z pliku y.

Tworzony jest 1 proces.  
cat y | x

Uruchamia procesy dla poleceń cat y i x.

Standardowe wyjście polecenia cat y (czyli zawartość pliku y) jest przekierowywane do standardowego wejścia polecenia x.

Tworzone są 2 procesy jednocześnie.  
x & y

Uruchamia proces dla polecenia x w tle (asynchronicznie).

Następnie uruchamia proces dla polecenia y.

Tworzone są 2 procesy, ale nie jednocześnie (najpierw x, potem y).  
x && y

Uruchamia proces dla polecenia x.

Jeśli x zakończy się sukcesem (kod wyjścia 0), następnie uruchamia proces dla polecenia y.

Tworzone są 1 lub 2 procesy, w zależności od powodzenia x   
x || y

Uruchamia proces dla polecenia x.

Jeśli x zakończy się niepowodzeniem (kod wyjścia inny niż 0), następnie uruchamia proces dla polecenia y.

Tworzone są 1 lub 2 procesy, w zależności od powodzenia x   
x ; y &

Uruchamia proces dla polecenia x.

Po zakończeniu x, uruchamia proces dla polecenia y w tle.

Tworzone są 2 procesy, ale nie jednocześnie (najpierw x, potem y).  
cat x > y < z

Uruchamia proces dla polecenia cat x.

Standardowe wejście dla cat jest przekierowywane z pliku z, ale ponieważ cat x już używa pliku x jako wejścia, przekierowanie < z jest ignorowane.

Standardowe wyjście polecenia cat x jest przekierowywane do pliku y.

Tworzony jest 1 proces.

Zadanie  
----------  
Załóżmy, że plik /etc/passwd miałby strukturę  
  
login:id:gr0:shell:gr1,gr2,g3  
  
tnowak:1001:tnowak:/bin/bash:admin,mail,www  
  
przy czym gr0 występuje zawsze, a dodatkowe nazwy grup po ostatnim dwukropku  
są opcjonalne.  
  
Napisz skrypt 'gr', który jako argumenty pobiera nazwy użytkowników, a następnie  
dla każdego użytkownika w osobnych liniach wypisuje "user: " i nazwę użytkownika, a potem po jednej nazwie grupy w linii, przykładowo:  
  
$ ./gr tnowak  
user: tnowak  
tnowak  
admin  
mail  
www  
$

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie