Procesy

Proces (process) – program będący w trakcie wykonywania
- systemowe
- użytkownika

System wsadowy wykonuje zadania (jobs) , system z podziałem czasu wykonuje procesy użytkownika(user programs) lub prace (jobs)

Sekcja tekstu(text section) – kod programu
licznik rozkazów (program couter) – adres następnego rozkazu
zawartość rejestrów procesora
stos procesu(process stack) – tymczasowe dane procesu (parametry metod, zmienne lokalne)
sekcja danych (data section) – zmienne globalne

Program nie jest procesem. Program - pasywny obiekt na dysku, proces - aktywnie wykonywany

Stan procesu

nowy – proces został utworzony
aktywny – procesor wykonuje rozkazy tego procesu
czekający – proces czeka na zdarzenie (np. na zakończenie operacji wejścia – wyjścia)
gotowy – proces czeka na przydział procesora
zakończony – proces zakończył działanie

Blok kontrolny procesu

Blok kontrolny procesu (process control block , PCB), blok kontrolny zadania -
stan procesu
licznik rozkazów
rejestry procesora : akumulatory , rejestry indeksowe, wskaźniki stosu, rejestry ogólnego przeznaczenie, rejestry warunków,
informacje o planowaniu przydziału procesora : priorytet, wskaźniki do kolejek planowania
informacje o zarządzaniu pamięcią : np zawartości rejestrów granicznych , tablice stron, tablice segmentów
informacje do rozliczeń – ilość zużytego czasu procesora, czasu rzeczywistego, ograniczenia czasowe, numery kont, zadań, procesów
informacje o stanie wejścia wyjścia – wykaz otwartych plików i przydzielonych procesowi urządzeń wejścia wyjścia

Proces może mieć wiele wątków, wykonujących różne zadania

Jeżeli w systemie jest wiele procesów, wskazane jest żeby często następowały pomiędzy nimi przełączenia, żeby użytkownik mógł interakcyjnie współpracować z każdym procesem.

Kolejki planowania

kolejki zadań (job queue) – kolejka ze wszytkimi procesami w systemie
kolejka procesów gotowych (ready queue) – kolejka z procesami, które nie czekają na żadne zdarzenie. Zazwyczaj ma postać listy powiązanej jednokierunkowej ze wskaźnikami na początek i koniec.

Kolejka do urządzenia (device queue) - kolejka procesów wymagających zdarzenia wykonanego przez dane urządzenie do kontynuacji działania.

Wyekspediowany proces (dispatched) – proces który przenosi się z kolejki gotowych na procesor.
Po przydziale procesora proces może zostać usunięty z procesora do odpowiednej kolejki z powodów : zapotrzebowania na operacje we-wy (kolejka oczekujących) , utworzenia podprocesu (kolejka oczekujących) i czekania na jego zakończenie, przerwania (kolejka gotowych)

Po zakończeniu procesu jest on usuwany ze wszystkich kolejek i zwalnia się pamięć jego bloku kontrolnego i wszystkie przydzielone mu zasoby.

Planiści

Planista (scheduler), program szeregujący – program wybierający w jakiej kolejności procesy są wykonywane
Planista długoterminowy (long-term scheduler) , planista zadań (job scheduler) - wybiera proces z puli procesów w pamięci masowej i ładuje je do pamięci operacyjnej. Nadzoruje stopień wieloprogramowości (liczbę procesów w pamięci). Nie we wszystkich systemach występuje. Jak go nie ma pamięć operacyjna znacząco może ograniczać płynność pracy systemu.
Planista krótkoterminowy (short-term scheduler), planista przydziału procesora (CPU scheduler) - wybiera, który z gotowych procesów jest ekspediowane na procesor. Podejmuje decyzje co najmniej raz na 100 ms.
Planista średnioterminowy (medium-term scheduler) – usuwa z pamięci operacyjnej procesu i przenosi je do masowej, ale z zapisaniem danych potrzebnych do wznowienia ich. Proces ten to wymiana (swapping)

Proces ograniczony przez we-w (I/O bound process) - proces który większość czasu spędza na czekaniu na operacje we – wy – w kolejkach urządzeń

Proces ograniczony przez procesor (CPU bound process) – proces który większość czasu spędza na wykonywaniu obliczeń na procesorze i rzadko wykonuje operacje we-wy – w kolejce gotowych

Planista długoterminowy musi dobrze dobrać mieszankę procesów (process mix), żeby żaden z rodzajów nie dominował.

Przełączanie kontekstu(context switch) – przechowanie danych starego procesu i załadowanie nowych
Wartość czasu przełączania kontekstu zależy od maszyny od 1 do 1000 micro s. Jeżeli jest wiele zbiorów rejestrów można przełączać jedynie, który zbiór jest aktualnie rozpatrywany. Przełączanie kontekstu to wąskie gardło wieloprocesowości, którego stara się unikać.

Działania na procesach

Tworzenie procesu – wywołanie sysytemowe create-process (w unixie fork) , proces macierzysty/proces rodzicielski/przodek (parent process) tworzy procesy potomków (childern).
Tworzenie procesów ma strukturę drzewiastą.

Procesy potomne mogą dostawać swoje zasoby, lub korzystać z jakiegoś podzbioru zasobów procesu macierzystego. Mogą też działać współbierznie z procesem macierzystym lub proces macierzysty może

oczekiwać na zakończenie, części lub wszystkich swoich podprocesów. Proces potomny może być kopią procesu macierzystego lub nowym procesem.

Wywołanie execlp – ładuje plik binarny do pamięci niszcząc dotychczasowy obraz procesu wywołującego i wykonuje plik binarny,
Identyfikator procesu (Process identifier)

Kończenie procesu – wywołanie exit, wszystkie zasoby zostają odebrane przez SO. Przyczyny:
- Wykonanie ostatniego rozkazu,
- Inny proces (zwykle tylko proces macierzysty) zakończył nasz proces wywołaniem systemowym abort
    * potomek nadużył któregoś z przydzielonych mu zasobów
    * wykonanie się potomka stało się zbędne,
    * proces macierzysty się kończy i system zabija potomków – kończenie kaskadowe (cascading termination)

Procesy współpracujące
Proces niezależny (independent process) – nie może oddziaływać na inne procesy a one na niego (nie dzieli danych)
Proces współpracujący (cooperating) – dzieli dane
Powody współpracy procesów :
- Dzielenie informacji
- Przyspieszanie obliczeń (zrównoleglanie)
- Modularność (oddzielanie różnych funkcji na różne procesy)
- Wygoda (równoległe wykonywanie wielu zadań korzystających z tych samych zasobów)

Bufor pamięci może być udostępniany za pośrednictwem komunikacji międzyprocesowej (inter-process-communication/ IPC)

System przekazywania komunikatów
Komunikaty mają stałą lub zmienną długość. Pomiędzy dwoma procesami do komunikacji musi istnieć łącze komunikacyjne (communication link). Realizowane przez send i receive. Do komunikacji potrzebne jest spójne nazewnictwo procesów (adresowanie komunikatu)

Komunikacja bezpośrednia – send(adresat, komunikat) , receive(nadawca/ _, komunikat)
- dane łącze dotyczy tylko 2 konkretnych procesów
- łącze ustawiane automatycznie pomiędzy parą procesów które mają się ze sobą komunikować
- między każdą parą procesów istnieje dokładnie jedno łącze
- obiorca nie we wszystkich modelach musi znać nadawce

Komunikacja pośrednia – komunikaty przez skrzynki pocztowe (mail – boxes)/ porty (ports)
- łącze jak procesy dzielą skrzynkę
- skrzynka może łączyć więcej niż 2 procesy
- jeden proces może być połączony z więcej niż jedną skrzynką.

Jeśli wiele procesów odbiera z danej skrzynki można :
- zezwalać na łącza tylko między dwoma procesami w danym momencie
- pozwolenie w danej chwili tylko jednemu procesowi na wykonanie operacji receive
- dopuścić aby losowy proces odebrał komunikat
- utorzsamić skrzynkę z właścicielem (jak kończy się właściciel to niszczy się skrzynka).

Skrzynka może należeć do procesu lub do systemu.

Przekazywane z blokowaniem = synchroniczne (synchronous)
proces nadawczy zostaje zablokowany aż jego komunikat nie zostaje odebrany
proces odbiorczy jest zablokowany aż nie nadejdzie jego komunikat
moment nadania to spotkanie (randezvous)
Przekazywanie bez blokowania = asynchroniczne (asynchronous)
proces nadawczy od razu po nadaniu wznawia działanie
odbiorca otrzymuje poprawny komunikat lub nie i wznawia działanie

Buforowanie komunikatów
pojemność zerowa – nadawca czeka na odbiór poprzedniego komunikatu – komunikaty bez buforowania
pojemność ograniczona – kolejkuje się n komunikatów
pojemność nieograniczona – nadawca nigdy nie jest blokowany i wszystko jest kolejkowane

Mach

Większość wywołań systemowych i komunikacja między zadaniami odbywa się przez komunikaty. Wywołanie msg_rpc uaktywnia zdalną procedurę RPC, wysyłając komunikat i oczekując na komunikat zwrotny. Przy stworzeniu każdego zadania powstają 2 skrzynki : zawiadomień i jądra. Wiadomości przychodzące do skrzynek są kolejkowane metodą FIFO (max 8 elementów). Każda skrzynka ma właściciela i w danym momencie tylko jeden proces może odbierać komunikaty z niej. Komunikat ma stałej wielkości nagłówek zawierający długość całego komunikatu i zmiennej długości porcje danych (każda ma swój rozmiar, typ i wartość). Jeżeli wątek nadaje do pełnej skrzynki lub oczekuje na komunikat z pustej, może czekać aż się zwolni miejsce, czekać maksymalnie n sekund, wrócić do pracy, czasowo samemu zachować komunikat i wysłać później. Zbiór skrzynek (mailbox set) to grupa skrzynek która przez dany proces jest traktowany jak jedna skrzynka. Proces może odbierać komunikaty tylko ze skrzynek lub grup do których ma prawo odbioru.

Windows 2000

Organizacja przekazywania komunikatów w tym systemie nosi nazwę wywołania procedury lokalnej (LPC, local procedure call). Komunikacja przebiega za pomocą portów połączeń i komunikacyjnych w następujący sposób :
- Klient tworzy uchwyt do podsystemowego obiektu portu połączenia
- Klient wysyła żądanie połączenia
- Serwer tworzy 2 prywatne porty i przekazuje klientowi uchwyt do jednego z nich
- Klient i serwer posługują się odpowiednimi uchwytami portowymi do wysyłania komunikatów lub przywołań (callbacks) i oczekiwania na odpowiedzi

Krótkie komunikaty są przekazywane przez porty a długie przy pomocy obiektu sekcji (pamięci dzielonej)

Komunikacja klient serwer

Gniazdo (socket) – punkt końcowy komunikacji. Identyfikuje się za pomocą adresu IP skonkatenowanego z numerem portu. Serwer nasłuchuje na odpowiednich portach. Jeżeli klient zarząda serwer może mu wydzielić osobny port. Każde połączenia muszą być niepowtarzalne dlatego wielu klientom nie przydziela się tego samego portu. Jest to popularny i wydajny, ale można przesyłać tylko pozbawione struktury strumienie bajtów.

W Javie są 3 różne typy gniazd : połączeniowe (connection-oriented sockets w protokole TCP) - klasa Socket, bezpołączeniowe (connectionless sockets w protokole UDP) – klasa DtatagramSocket, i klasa Multicast Socket. Gniazdo umożliwia rozsyłanie danych na wielu odbiorców.
Serwer czasu dobowego(time-of-day server)
komputer lokalny (localhost)

Zdalne wywołanie procedur RPC
Komunikaty są adresowane do demona RPC prowadzącego nasłuch portu RPC na odległym systemie i zawierają identyfikator funcji oraz parametry potrzebne do jej wywołania. Wynik funkcji zostanie odesłany z powrotem w oddzielnym komunikacie. Port jest numerem umieszczonym na początku pakietu z komunikatem. Semantyka RPC umożliwia wywołanie procedury zdalnej analogicznie jak lokalnie. System RPC udostępnia u klienta namiastkę (stub) procedury wywoływaną jak sama funkcja, która lokalizuje port na serwerze i przetacza (marshalls) parametry. Analogiczna namiastka w serwerze odbiera komunikat. Wiele systemów RPC definiuje niezależną od maszyny zewnętrzną reprezentację danych (external data representation, XDR), i konwersja następuje przed wysłaniem i po odebraniu komunikatu, dzięki czemu serwer i klient mogą mieć różne reprezentacje danych i nadal się komunikować. Każdy komunikat jest zaopatrzony w znacznik czasu, aby nie obsługiwać wielokrotnie tego samego zapytania przez problemy na łączach.
Czasami przy kompilacji identyfikator funkcji może być zastąpiony adresem w pamięci co może wywołać problemy przy zdalnym wywołaniu. Aby sobie z tym poradzić można zamienić identyfiktor na numer portu jeżeli serwer go nie zmieni, lub można wywoływać demona spotkań/swata (matchmaker), który zdobywa od serwera aktualny numer portu.
Przy pomocy tego schematu można implementować rozproszony system plików, gdzie zapytanie to operacja dyskowa, a odpowiedź zwrotna to dane wynikające z wywołania tej operacji.

Zdalne wywoływanie metod (remote method invocation RMI)
Własność systemu Java, za jego pomocą można wywłać metodę zdalnego obiektu. RMI może modyfikować zdalne obiekty za pomocą metody (a nie tylko wywołać procedury jak w RPC), a jako parametry można metodą przekazywać całe obiekty a nie zwykłą strukturą danych. Procedure realizuje się za pomocą namiastek i szkieletów. Namiaska zdalengo obiektu jest wywoływana u klienta i odpowiada za tworzenie paczek (parcel), zawierających nazwy i parametry metody do wywołania na serwerze. Szkielet przesyła informację zwrotną z serwera do klienta.
Jeżeli parametry są obiektami lokalnymi (nieodległymi, nonremote) to przekazuje się je przez skopiowanie, za pomocą szeregowania obiektu (object serialization), jak parametry są obiektami zdalnymi to przekazuje się je przez odniesienie. Parametry lokalne muszą implementować interfejs serializable