Zadanie 1.

*Wyjaśnij różnice między **powłoką (ang. shell), system operacyjnym i jądrem systemu operacyjnego** (ang. kernel). W tym celu dobierz kilka przykładów powszechnie wykorzystywanego oprogramowania. Jakie są główne zadania systemu operacyjnego z punktu widzenia programisty?*

Wyjaśnij różnice między **powłoką (ang. shell), system operacyjnym i jądrem systemu operacyjnego** (ang. kernel).

System operacyjny

- oprogramowanie zarządzające systemem komputerowym,
- tworzące środowisko do uruchamiania
- kontrola zadań.
- planowaniem oraz przydziałem czasu procesora poszczególnym zadaniom,
- kontrolą i przydziałem pamięci operacyjnej dla uruchomionych zadań,
- dostarczaniem mechanizmów do synchronizacji zadań i komunikacji pomiędzy zadaniami,
- obsługą sprzętu oraz zapewnieniem równolegle wykonywanym zadaniom jednolitego, wolnego od interferencji dostępu do sprzętu.
- w jego skład wchodzi:
    - powłoka
    - jądro
    - sterowniki - obsługa sprzętu

Shell

- pośrednik pomiędzy użytkownikiem a jądrem
- interpretuje jego polecenie i przesyła do SO
- nie jest integralną częścią systemu operacyjnego, można go zastąpić; Polesiuk mówi że jest
- CLI, GUI (CLI: CMD, Bash; GUI: GNOME, Windows Explorer)
- działa w trybie REPL (read-eval-print loop)
- zarządzanie plikami i uruchamianie aplikacji

Jądro

- rdzeń SO
- obsługa I/O
- zarządza czasem procesora
- ma kontrolę nad wszystkim w SO
- obsługa SYSCALL

W tym celu dobierz kilka przykładów powszechnie wykorzystywanego oprogramowania.

System Operacyjny

- Windows
- Linux
- MS-DOS

Shell

- CLI:
  - CMD
  - Bash
- GUI:
  - GNOME
  - Windows Explorer

Jądro

- Unix
- OpenBSD
- FreeBSD

Jakie są główne zadania systemu operacyjnego z punktu widzenia programisty?

- Dostarczanie środowiska do uruchamiania zadań
- Obsługa I/O

Zadanie 2.

Czym jest **zadanie** w **systemach wsadowych**? Jaką rolę pełni **monitor**? Na czym polega **planowanie zadań**? Zapoznaj się z rozdziałem „System Supervisor” dokumentu IBM 7090/7094 IBSYS Operating System2 . Wyjaśnij pobieżnie znaczenie poleceń **języka kontroli zadań** (ang. Job Control Language) użytych na rysunku 3 na stronie 13. Do jakich zastosowań używa się dziś systemów wsadowych?
Wskazówka: Bardzo popularnym systemem realizującym szeregowanie zadań wsadowych jest SLURM3 .

Czym jest **zadanie** w **systemach wsadowych**? Jaką rolę pełni **monitor**? Na czym polega **planowanie zadań**?

System wsadowy - system operacyjny dla pierwszych komputerów, charakteryzujący się wsadowym przetwarzaniem zadań oraz tym, że użytkownicy nie mieli bezpośredniego dostępu do komputera, a jedynie poprzez operatora. Zadania od poszczególnych użytkowników gromadzone są na nośniku jako wsad i wykonywane jedno po drugim w określonej kolejności.
Zadanie - zbiór instrukcji przetwarzanych w określonej kolejności, najczęściej bez ingerencji użytkownika.
Monitor - program na stałe przebywający w pamięci operacyjnej, umożliwia automatyczne przekazywanie sterowania pomiędzy uruchomionymi programami i śledzi ich przebieg realizacji
Planowanie zadań - polega na decydowaniu o kolejności uruchamiania zadań tak aby było jak najoptymalniej. Kryteria to:

- Fair-share (how over- or under-served a user/group is)
- Age (how long queued)
- Size (favor larger or smaller jobs)
- Queue/partition priority factor
- Quality Of Service (QOS) priority factor

Wyjaśnij pobieżnie znaczenie poleceń **języka kontroli zadań** (ang. Job Control Language) użytych na rysunku 3 na stronie 13.

Job Control Language (JCL) – język opisu zadań, jest zbiorem wyrażeń (poleceń), które są przekazywane do systemu aby wykonać program podążając według pewnych instrukcji wyjściowych i wejściowych. Takie wyrażenia przekazują systemowi, gdzie znajdują się odpowiednie wejścia i jak należy przetworzyć owo wejście (uruchomienie programu) i co z rezultatem działania programu. Job pozwala na wykonanie zadania (zadań) w tle pracy systemu.

Najważniejszymi wyrażeniami z powyższych klas są: JOB, EXEC, DD.

JOB - to wyrażenie musi znajdować się na początku każdego job i określa przetwarzane informacje.
EXEC (EXECUTE) - to wyrażenie musi występować na początku każdego *step*, definiuje jaki program lub procedurę należy uruchomić, dostarcza parametrów. W każdym *job* może znajdować się maksymalnie 255 *step*.
DD (Data Description) - wyrażenie opisujące wejścia/wyjścia (input/output) data sets i ich własności. Wyrażenie DD zazwyczaj występuje po wyrażeniu EXEC.

$RELEASE Card

Format:

1	16
$RELEASE	SYSxxx

This card causes the unit assigned to the specified system unit function to be released from the function. If the unit was concurrently assigned to other system unit functions, it remains assigned to those functions.

$IBEDT

Upon recognizing this card, the System Supervisor calls the System Editor into core storage from a System Library Unit and relinquishes control to it. The same effect may be accomplished by a SEXECUTE card with the name EDITOR specified in the variable field. The control cards that are recognized by the System Editor are described in the section "System Editor."

$IBSYS

When this card is read by a subsystem or by the System Editor, the System Supervisor is called into core storage and control is relinquished to it. The System Supervisor then reads and processes succeeding control cards until control is relinquished to a subsystem by means of a $EXECUTE card or to the System Editor by means of a $EXECUTE EDITOR or $IBEDT card.

http://bitsavers.org/pdf/ibm/7090/C28-6248-7_v13_IBSYS_Dec66.pdf

Do jakich zastosowań używa się dziś systemów wsadowych?

- In the high-performance computing environment, **burst buffer** is a fast and intermediate storage layer positioned between the front-end computing processes and the back-end storage systems. (pośrednik między obliczeniami a systemem plików)
- Job arrays are groups of jobs with the same executable and resource requirements, but different input files.

Zadanie 3.

*Jaka była motywacja do wprowadzenia **wieloprogramowych** systemów wsadowych? W jaki sposób wieloprogramowe systemy wsadowe wyewoluowały w systemy z **podziałem czasu** (ang. time-sharing)? Podaj przykład historycznego systemu **interaktywnego**, który nie jest wieloprogramowy.*

Jaka była motywacja do wprowadzenia **wieloprogramowych** systemów wsadowych?

system wsadowy - system operacyjny dla pierwszych komputerów, charakteryzujący się wsadowym przetwarzaniem zadań oraz tym, że użytkownicy nie mieli bezpośredniego dostępu do komputera, a jedynie poprzez operatora.

Motywacja: Jak Efektywniej wy

Pierwsze komputery były wielkimi maszynami wyposażonymi najczęściej w drukarki, czytniki kart, przewijaki taśm i konsolę. Ze względu na ich cenę i koszty utrzymania priorytetem było efektywne wykorzystanie czasu pracy komputera. Użytkownicy nie mieli bezpośredniego dostępu do komputera. Pracę komputera nadzorował wyspecjalizowany operator. W momencie gdy pojawiły się dyski magnetyczne, nowy szybki nośnik informacji, powstała nowa technologia, pozwalająca na zwiększenie wydajność systemu: *spooling*

Spooling polega na tym, że równocześnie odbywają się trzy rzeczy:

- nowe zadania są wczytywane przez czytnik(i) kart i zapisywane na dysku,
- procesor pobiera kolejne zadania z dysku, wykonuje je i zapisuje wyniki na dysku,
- wyniki zakończonych zadań są drukowane na drukarce(-kach).

Dzięki spoolingowi operacje wejścia/wyjścia mogły odbywać się równocześnie z obliczeniami. Za cenę dysku magnetycznego i części pamięci operacyjnej można było tak efektywnie wykorzystać czas pracy procesora. Synchronizacja wczytywania zadań, wypisywania wyników oraz zarządzanie informacjami magazynowanymi na dysku stało się nowym zadaniem systemu operacyjnego.

Mimo, iż spooling umożliwił wydajniejsze wykorzystanie procesora, to nadal procesor był momentami bezczynny, np. w trakcie wykonywania operacji dyskowych lub w trakcie korzystania z pomocniczych pamięci zewnętrznych, takich jak taśmy magnetyczne. Dalsze zwiększenie wydajności osiągnięto dzięki zastosowaniu ***wieloprogramowości*.**

Wieloprogramowe systemy wsadowe

- dopóki są jakieś zadania w pamięci do wykonania jednostka centralna nie jest bezczynna
- jeśli nie wszystkie zadania z puli zadań mogą zostać umieszczone w pamięci to system operacyjny musi wybierać spośród nich
- przechowywanie wielu zadań w pamięci wymaga mechanizmów zarządzania pamięcią
- jeśli kilka zadań w pamięci operacyjnej jest gotowych do działania to należy wybrać któryś z nich i przydzielić mu procesor
- systemy tego typu dużo są bardziej skomplikowane od prostych systemów wsadowych

Wieloprogramowość polega na tym, że w pamięci operacyjnej komputera znajduje się kilka działających programów. Taki program, który został uruchomiony i załadowany do pamięci operacyjnej będziemy nazywać *procesem*. (Jeżeli ten sam program został uruchomiony kilkakrotnie, to tworzy on kilka odrębnych procesów.) Gdy jeden z procesów czeka na zakończenie operacji wejścia/wyjścia, procesor nie musi być bezczynny, może wykonywać inny proces.
SPOOL - Simultaneous Peripheral Operation On-Line jednoczesna bezpośrednia praca urządzeń

W jaki sposób wieloprogramowe systemy wsadowe wyewoluowały w systemy z **podziałem czasu** (ang. time-sharing)?

Wieloprogramowość umożliwiła efektywne wykorzystanie czasu pracy komputera, jednak czas pracy programistów był wykorzystywany mało efektywnie. Nadal nie mieli oni bezpośredniej styczności z komputerem, a jedynie przetwarzali wsadowo zadania. Każdy błąd w programie powodował konieczność

powtórzenia zadania i wiązał się z długotrwałymi oczekiwaniami na kolejne wyniki. Nie było też mowy o śledzeniu wykonania programu.

Systemy z podziałem czasu były pierwszymi, które pozwoliły na *interaktywną* pracę wielu użytkowników z komputerem. Czytniki kart perforowanych zostały zastąpione terminalami, przy których mogli pracować użytkownicy. Podział czasu polega na tym, że procesor jest przydzielany procesom znajdującym się w pamięci komputera w małych porcjach, "kwantach". Gdy kwant czasu się kończy, procesor jest przydzielany kolejnemu oczekującemu procesowi. Kwanty czasu są na tyle małe, że użytkownicy mają wrażenie płynnej interakcji z komputerem. Jednocześnie, dzięki temu, że jeden użytkownik nie jest w stanie cały czas obciążać procesora, każdy z użytkowników ma wrażenie, że ma do dyspozycji dużą część mocy obliczeniowej komputera.

W systemie, w którym użytkownicy interakcyjnie komunikują się z procesami, czas obrotu nie jest dobrą miarą wydajności systemu. Lepszą miarą jest średni *czas reakcji*. Czas reakcji to czas, jaki upływa od wykonania przez użytkownika jakiejś akcji (naciśnięcia klawisza, wprowadzenia polecenia, kliknięcia myszą itp.) do zareagowania przez proces na tę akcję. Na czas reakcji składa się również czas oczekiwania przez proces na przydział procesora. Dlatego też, im krótszy jest kwant czasu, na jaki przydzielany jest procesor, tym krócej proces czeka na przydzielenie procesora.

Z chwilą, gdy użytkownicy mieli bezpośredni kontakt z systemem operacyjnym, pojawiły się nowe potrzeby i wzrosły wymagania użytkowników wobec środowiska pracy oferowanego przez system operacyjny. Jedna z takich potrzeb dotyczyła mechanizmu przechowywania programów i rozmaitych danych. W rezultacie powstały *systemy plików*. Realizacja systemu plików i operacji na plikach stała się kolejnym elementem systemu operacyjnego.

Przetwarzanie wsadowe było też łączone z podziałem czasu. Działo się tak np. w przypadku komputerów, które zostały skonstruowane z myślą o przetwarzaniu wsadowym, a następnie ich systemy operacyjne zostały rozszerzone o podział czasu. Przykładem może być tu system operacyjny IBM OS/360, do którego dodano opcję podziału czasu.

Podaj przykład historycznego systemu **interaktywnego**, który nie jest wieloprogramowy.

DOS
Pierwszy Unix z 1969

Zadanie 4

*Wymień mechanizmy sprzętowe niezbędne do implementacji **wywłaszczania** (ang. preemption). Jak działa **algorytm rotacyjny** (ang. round-robin)? Jakie zadania pełni **planista** (ang. scheduler) i **dyspozytor** (ang. dispatcher)? Który z nich realizuje **politykę**, a który **mechanizm**?*

Wymień mechanizmy sprzętowe niezbędne do implementacji **wywłaszczania** (ang. preemption)

Wywłaszczenie – technika używana w środowiskach wielozadaniowych, w której algorytm szeregujący zwany planistą lub dyspozytorem może wstrzymać aktualnie wykonywane zadanie (np. proces lub wątek), aby umożliwić działanie innemu zadaniu. Dzięki temu rozwiązaniu zawieszenie jednego procesu nie powoduje blokady całego systemu operacyjnego.

- **programmable interrupt controller** (**PIC**) - is a device that is used to combine several sources of interrupt onto one or more CPU lines, while allowing priority levels to be assigned to its interrupt outputs.
- timer interrupt
- procesor musi obsługiwać tryb user/supervisor (?)

Jak działa **algorytm rotacyjny** (ang. round-robin)?

Typowym algorytmem planowania wywłaszczającego jest **algorytm rotacyjny**. Każdy proces wykonywany jest co najwyżej przez pewien okres czasu, po czym następuje przełączenie kontekstu na inny proces. Po jakimś czasie nastąpi wznowienie procesu przerwanego. Proces może przed upływem kwantu czasu zgłosić żądanie zasobowe, zrezygnować dobrowolnie z procesora lub zakończyć się, co skutkuje przydzieleniem nowego kwantu dla następnego procesu. W planowaniu rotacyjnym wszystkie procesy mają ten sam priorytet. Zasadniczym kosztem stosowania algorytmu rotacyjnego jest zużycie czasu procesora na przełączanie kontekstu. Z punktu widzenia przetwarzania użytkowego czas ten jest marnowany.
TL;DR **Planowanie rotacyjne (ang. Round Robin, RR)** — po ustalonym kwancie czasu proces wykonywany jest przerywany i trafia do kolejki procesów gotowych.

Jakie zadania pełni **planista** (ang. scheduler) i **dyspozytor** (ang. dispatcher)?

- **dyspozytor (ang. dispatcher) -** the piece of code that is responsible for taking a task/process environment, making it current for a processor and start executing it… It is also responsible for setting bounds/limits to how long it will run and how much resources are consumed. The “Scheduler” (another piece of software) will then determine which task/process should be run next.
- **Scheduler** - Ogólnym zadaniem **planistów** (programów szeregujących) jest wybieranie procesów z pewnego zbioru tak, aby dążyć do optymalizacji przetwarzania w systemie.

Który z nich realizuje **politykę**, a który **mechanizm**?

Polityka:

Policies are ways to choose which activities to perform.

W systemie Linux, podobnie jak w innych współczesnych systemach operacyjnych (Windows 2000, współczesne implementacje systemów uniksopodobnych) uwzględniono szeregowanie procesów czasu rzeczywistego. Są to rozwiązania spełniające wymagania tzw. łagodnego (tolerancyjnego) czasu rzeczywistego, co znaczy, że system próbuje wykonywać je szybciej niż inne zadania, ale nie gwarantuje ich zakończenia w określonym czasie (przed linią krytyczną).

Priorytety dla zadań czasu rzeczywistego (czyli klas SCHED_FIFO i SCHED_RR) są zawsze wyższe — mają mniejsze wartości — niż priorytety zadań zwykłych.

- Stosowany jest algorytm rotacyjny z wywłaszczaniem, oparty na priorytetach dynamicznych.
- Wyróżnia 140 poziomów priorytetu związanych z 3 klasami (**politykami**) szeregowania:
    - SCHED_OTHER — polityka szereg, zwykłych zadań,
    - SCHED_FIFO — polityka szereg, zadań czasu rzeczywistego zgodnie z zasadą FIFO,
    - SCHED_RR — polityka szeregowania zadań czasu rzeczywistego w sposób rotacyjny.
- Większa wartość oznacza niższy priorytet.

Mechanizm:

Mechanisms are the implementations that enforce policies, and often depend to some extent on the hardware on which the operating system runs. For instance, a processes may be granted resources using the first come, first serve policy. This policy may be implemented using a queue of requests. Often the kernel provides mechanisms that are used to implement policies in servers.

dyspozytor mechanizm, planista politykę

Zadanie 5.

Zapoznaj się z podrozdziałem „The Elements of Operating-System Style” książki „The Art of Unix Programming”. Czemu system operacyjny powinien (a) umożliwiać szybkie tworzenie procesów i łatwą komunikację międzyprocesową (b) przechowywać dane w plikach tekstowych, a nie binarnych (c) udostępniać szereg narzędzi programistycznych (d) oferować bogaty wybór programów działających w linii poleceń?

(a) umożliwiać szybkie tworzenie procesów i łatwą komunikację międzyprocesową

OS powinien umożliwiać szybkie tworzenie procesów, aby nie tworzyć procesów potworów, dużych nieefektywnych programów i by było wiadomo, co dlaczego się dzieje. Również powinien umożliwiać szybkie tworzenie procesów, gdyż jeśli ich nie tworzy, to program musi używać wielowątkowości, a mógłby używać kilku prostych procesów. Zrozumienie i używanie asynchronicznego wejścia/wyjścia jest konieczne. Do procesów tworzony jest wielowarstwowy kod w c++ zamiast relatywnie prostej i płaskiej hierarchii w C. Procesy powinny łatwo komunikować się, gdyż bez łatwej komunikacji będą wykorzystywane mechanizmy, które są brzydkie, nieeleganckie, nieefektywne i niebezpieczne (np. pliki tymczasowe) or by knowing far too much about each others' implementations.

(b) przechowywać dane w plikach tekstowych, a nie binarnych

OS powinien przechowywać dane w plikach tekstowych, a nie w plikach binarnych, gdyż format pliku binarnego nie jest czytelny. Do przeczytania pliku binarnego potrzebne są specjalne narzędzia, deweloperzy będą się skupiać na narzędziach zamiast na treści, różne formaty plików mogą nie być kompatybilne. Warto także wspomnieć, że otwieranie binarek z nieznanych źródeł jest niebezpieczne i niepolecane.

(c) udostępniać szereg narzędzi programistycznych

OS powinien udostępniać szereg narzędzi programistycznych, aby wesprzeć hobbistyczne zainteresowania użytkownika software developmentem, pozwalać mu się rozwijać i pisać zwykłe programy, a jednocześnie aby development mógł być tani, by nie wymagał ogromnych kosztów ze strony użytkownika i by koszty narzędzi programistycznych nie były barierą w rozwoju i programowaniu.

(d) oferować bogaty wybór programów działających w linii poleceń?

OS powinien oferować bogaty wybór programów działających w linii poleceń, aby wszystkie programy były ze sobą kompatybilne i by współpracowały ze sobą, by outputy mogły być wykorzystywane jako inputy, aby zdalna obsługa i administracja systemem była prostsza i wygodniejsza, wspierana i nie obciążająca w znaczny sposób sieci, aby wszystkie skrypty były ze sobą kompatybilne, możliwe do uruchomienia, aby daemony, serwery i procesy w tle były łatwe i wygodne do napisania.