L1.Z1. Wyjaśnij różnice między **powłoką** (ang. *shell*), **systemem 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?

powłoka - interpreter poleceń użytkownika kierowanych do kernela

- jest podstawowym interfejsem pomiędzy userem a sysopkiem
- powłoka, bo jest nakładką na kernel
- nie jest częścią systemu operacyjnego
- powłoka wykorzystuje emulator terminala jako std IO
- przykłady powłok linuksowych: sh shell i bash shell (domyślna powłoka w niektórych dystrybucjach linuksa)
- dwa rodzaje:
    - powłoka tekstowa - interpreter poleceń uruchamiany w trybie tekstowym, potocznie zwany konsolą
    - powłoka graficzna - ma zwykle postać menadżera plików, kontrolowana za pomocą myszy. Eksplorator – domyślna powłoka systemu MS Windows

system operacyjny - oprogramowanie działające w trybie jądra
SO spełniają dwie niepowiązane ze sobą funkcje:

- dostarczają programistom aplikacji (i programom aplikacyjnym) czytelnego, abstrakcyjnego zbioru zasobów będących odpowiednikami sprzętu
- zarządzają tymi zasobami sprzętowymi
    - pamięć (fizyczna, tablica stron)
    - procesor (translacja adresów, przerwania, cache, TLB)
    - urządzenia IO
    - kanały DMA

Sysopek jako rozszerzona maszyna:
Architektura większości komputerów na poziomie języka maszynowego (zestaw instrukcji, organizacja pamięci, IO i struktura magistral) jest prymitywna i niewygodna do programowania. Jednym z głównych zadań systemu operacyjnego jest ukrywanie sprzętu i dostarczanie programom (programistom) wygodnych, czytelnych, eleganckich i spójnych abstrakcji do wykorzystania (np. wykonywanie operacji IO na dysku – nie trzeba na poziomie sprzętowym, system operacyjny zawiera sterowniki dysku, które zapewniają interfejs do odczytu i zapisu bloków na dysku bez wchodzenia w szczegóły).

Prawdziwymi klientami systemu operacyjnego są programy aplikacyjne. Użytkownicy posługują się abstrakcjami dostarczanymi przez interfejs użytkownika – powłokę wiersza polecenia lub interfejs graficzny.

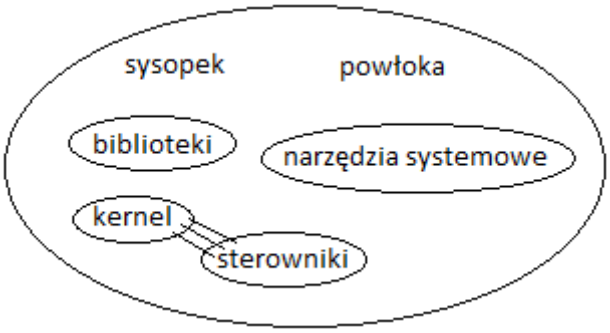
Sysopek jako menedżer zasobów:
Zadaniem sysopka jest zapewnienie uporządkowanego i kontrolowanego przydziału procesorów, pamięci i urządzeń IO pomiędzy różne programy rywalizujące o te zasoby; śledzenie informacji na temat tego, które programy korzystają z jakich zasobów

jądro systemu operacyjnego (kernel)

- znajduje się bezpośrednio nad warstwą sprzętową
- udostępnia interfejs do korzystania z hardware w postaci wywołań systemowych – syscalli (wywołania te mogą być wołane z poziomu usera, kiedy wymagane są działania w kernel mode – dostęp do wszystkich zasobów)
- dostarcza środowiska uruchomieniowego dla programów

Jakie są główne zadania systemu operacyjnego z punktu widzenia programisty?

- zarządza pamięcią (przydzielanie jej, ochrona dostępu do jej fragmentów, pamięć wirtualna)
- zapewnia ładną abstrakcję zasobów i zarządza nimi efektywnie
- zapewnia interakcję z userem



L1.Z2. Na podstawie dokumentacji wymień składowe **pakietu deb** ze szczególnym uwzględnieniem zawartości pliku `control`. Porównaj zarządzanie zainstalowanym oprogramowaniem z użyciem pakietów i instalatorów znanych z systemów nieuniksowych. Weź pod uwagę proces pobierania, weryfikacji, instalacji, konfiguracji i odinstalowania oprogramowania.

pakiet - skompresowane archiwum zawierające oprogramowanie, umożliwiające jego łatwą oraz szybką instalację, aktualizację lub dezinstalację

Wymień składowe pakietu deb ze szczególnym uwzględnieniem zawartości pliku control

Przykłady plików **deb** można znaleźć w `/var/cache/archives/`.

```
dpkg-deb -I package.deb - show control
dpkg-deb -c package.deb - list files which will be installed
ar tv package.deb       - list content
ar x package.deb        - extract files
```

Zbadajmy przykładowo plik parted_1.4.24-4_i386.deb:

```
$ ar tv parted_1.4.24-4_i386.deb
rw-r--r-- 0/0      4 Mar 28 13:46 2002 debian-binary
rw-r--r-- 0/0   1386 Mar 28 13:46 2002 control.tar.gz
rw-r--r-- 0/0  39772 Mar 28 13:46 2002 data.tar.gz
```

- `debian-binary` – zawiera informacje o wersji formatu deb - "2.0\n"
- `control.tar.gz` - zawiera cztery pliki:
  - control - zawiera dodatkowe informacje o pakiecie
  - md5sums - sumy kontrolne dla każdego pliku w data.tar.gz
  - postinst, prerm - zajmują się usuwaniem starych documentation files i dodawaniem linków z doc do share/doc
- `data.tar.gz` - archiwum zawierające wszystkie pliki, które muszą być zainstalowane wraz z ich ścieżkami; musi być ostatnim plikiem w archiwum deb

```
$ cat control
Package: parted
Version: 1.4.24-4
Section: admin
Priority: optional
Architecture: i386
Depends: e2fsprogs (>= 1.27-2), libc6 (>= 2.2.4-4), libncurses5 (>= \
5.2.20020112a-1), libparted1.4 (>= 1.4.13+14pre1), libreadline4 (>= \
4.2a-4), libuuid1
Suggests: parted-doc
Conflicts: fsresize
Replaces: fsresize
Installed-Size: 76
Maintainer: Timshel Knoll <timshel@debian.org>
Description: blablabla
```

- pola obowiązkowe:
  - package - nazwa pakietu; po tej nazwie inne pakiety będą się odnosić do tego pakietu w dependencjach
  - version - wersja pakietu; wersja pakietu, ma określony format i porządek - określenie minimalnej wersji w dependencjach
  - maintainer - email osoby, która stworzyła pakiet lub odpowiada za jego utrzymanie (kontakt w razie problemów z pakietem)
  - description - opis pakietu

- pola rekomendowane:
  - section - „typ" pakietu, np. utils, net, mail, text, x11, etc.
  - priority – określa priorytet pakietu w stosunku do systemu jako całości, np. required, standard, optional, extra
  - essential - (tak/nie) określa, czy pakiet jest wymagany do prawidłowego działania systemu, **dpkg** nie pozwoli na usunięcie, jeśli pakiet jest essential
  - architecture – architektura, na którą pakiet jest zbudowany, lub „all", jeśli jest niezależny od architektury
  - suggests - inne przydatne pakiety powiązane z tym
  - homepage - strona domowa oprogramowania dostarczanego z pakietem
  - depends – pakiety, które są wymagane do zainstalowania tego pakietu
  - replaces – pakiety, które ten zastępuje; pozwala nadpisać pliki innego pakietu
  - conflicts – pakiety, z którymi ten konfliktuje, np. zawierając pliki o tej samej nazwie; menedżer pakietów nie pozwoli, żeby konfliktujące pliki zainstalowały się jednocześnie

Porównaj zarządzanie zainstalowanym oprogramowaniem z użyciem pakietów i instalatorów znanych z systemów nieuniksowych

- Unix
  - pobieranie: pakiet z repozytorium pakietów, pobieranie tylko niezainstalowane zależności
  - weryfikacja: pakiety zweryfikowane przez twórców dystrybucji. + MD5
  - instalacja: instalacja pakietu i niezainstalowanych zależności, pakiet “wie”, jak się zainstalować
  - odinstalowanie: pakiet “wie”, jak się odinstalować
- Android-style
  - pobieranie: "apka" ze sklepu, wszystkie zależności są pobierane
  - weryfikacja: apki zweryfikowane przez właściciela sklepu (Google, Amazon, Samsung itd.) + aplikacje podpisane certyfikatem + SHA-256
  - instalacja: menedżer umie zainstalować apkę
  - odinstalowanie: menedżer umie odinstalować apkę
- Windows-style
  - pobieranie: brak zcentralizowanego systemu dystrybucji, wszystkie zależności zawarte w instalatorze lub użytkownik sam musi zadbać o ich spełnienie
  - weryfikacja: programy zweryfikowane przez użytkownika
  - instalacja: użytkownik podejmuje decyzje podczas procesu instalacji
  - odinstalowanie: użytkownik uruchamia aplikacje która wie, jak odinstalować program

Źródła: składowe pakietu, szczegóły control

rys historyczny

Pierwsza generacja

- maszyny prymitywne, wykonanie najprostszych obliczeń zajmowało kilka sekund
- ta sama grupa osób zajmowała się projektowanie, budową, programowaniem, obsługą i konserwacją każdej maszyny
- programowanie wyłącznie w języku maszynowym lub poprzez tworzenie obwodów elektrycznych łączonych za pomocą tysięcy kabli na specjalnych tablicach programowych

Tryb działania programisty:

- zapisanie się na wiszącej na ścianie liście zgłoszeń w celu uzyskania bloku czasu
- pójście do pokoju z maszyną i włączenie swojej tablicy programowej do komputera

Dzięki wprowadzeniu kart perforowanych na początku lat pięćdziesiątych procedura uległa usprawnieniom – zamiast używać tablic programowych, można było teraz pisać programy na kartach i odczytywać je z nich.

Druga generacja

- wprowadzenie tranzystorów, komputery stały się mniej zawodne (wcześniej 20 tysięcy lamp elektronowych, każda mogła się spalić w trakcie obliczeń)
- podział personelu ze względu na pełnione funkcje (projektant, konstruktor, operator itd.)

Proces uruchomienia zadania:

- programista pisał program na papierze w języku Fortran lub w asemblerze, dziurkował go na kartkach
- przynosił zbiór kart do pokoju wprowadzania danych, przekazywał operatorowi
- kiedy komputer skończył wykonywanie zadania, operator szedł do drukarki, oddzierał wynik działania programu i zanosił do pokoju wyników, skąd programista go później odbierał
- operator brał jeden z zestawów kart, które zostały przyniesione do pokoju wprowadzania danych i wczytywał go
- jeśli był potrzebny kompilator Fortrana, operator brał go z szafy i wczytywał do komputera

Obserwacja: dużo czasu marnotrawiono na chodzenie po pokoju komputerowym

Systemy wsadowe:

- IBM 1401 – czytanie kart, kopiowanie taśm, drukowanie wyników; IBM 7094 – obliczenia
- programista przynosi karty do 1401 (czytnik kart)
- 1401 wczytuje plik zadań na taśmę magnetyczną, którą trzeba było przenieść do pokoju komputerowego
- tam taśmę montowano w napędzie taśm
- operator ładował specjalny program („sysopek”), który odczytywał pierwsze zadanie z taśmy i je uruchamiał, 7094 wykonuje obliczenia, po zakończeniu każdego z zadań sysopek automatycznie wczytywał następne zadanie z taśmy i zaczynał je uruchamiać
- zamiast drukowania wynik był zapisywany na drugiej taśmie
- po zakończeniu przetwarzania wsadu operator wyjmował taśmy wejściową i wyjściową, wymieniał taśmę wejściową na następny wsad i przynosił taśmę wyjściową do komputera 1401 w celu wydrukowania go

Trzecia generacja

- problem: na początku lat 60 większość producentów utrzymywała po dwie, niezgodne ze sobą linie produktów: komputery naukowe (takie jak 7094) oraz komputery znakowe (takie jak 1401)
- droga sprawa, poza tym użytkownicy chcieli dużej maszyny, która byłaby zdolna do uruchamiania ich starych programów, tyle że szybciej
- IBM próbuje rozwiązać ten problem, produkuje serię System/360 – pierwsza linia komputerów, w której zastosowano układy scalone
- problem: największa zaleta tych komputerów to jednocześnie największa wada – komputery musiały być wydajne w wielu różnych zastosowaniach; dużo kolidujących ze sobą wymagań – wielki kod, dużo błędów, dużo nowych wersji poprawiających stare (ale też zawierające błędy)
- OS/360 i mu podobne spopularyzowały wieloprogramowość, której nie było w drugiej generacji

L1.Z3. 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 System. Wyjaśnij 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 SLURM .

Czym jest zadanie w systemach wsadowych? Jaką rolę pełni monitor? Na czym polega planowanie zadań?
zadanie - program lub zbiór programów, który dla danych wejściowych produkował dane wyjściowe (rozp. Od karty $JOB)

system wsadowy – idea ich działania polegała na pobraniu pełnego zasobnika zadań w pokoju wprowadzania danych i zapisaniu ich na taśmie magnetycznej za pomocą mniejszego komputera (IBM 1401 czytał karty, kopiował taśmy i drukował wyniki, ale nie nadawał się do obliczeń). Do obliczeń wykorzystywano większe i droższe komputery, np. IBM 7094.

monitor – protoplasta systemu operacyjnego; odczytywał kolejne zadania z taśmy wejściowej (interpretował control language) i je uruchamiał (eliminowało to konieczność obsługi pojedynczych programów przez człowieka). W monitorze były sterowniki, IO, zegar systemowy, ochrona pamięci.

planowanie zadań – ustalanie kolejności wykonywanych zadań. Na początku do komputera wrzucało się „paczkę z zadaniami", a jego planowanie to było zwykłe FIFO. Potem wprowadzono sortowanie np. na podstawie priorytetów zapisanych w JCL.

Wyjaśnij znaczenie poleceń języka kontroli zadań
język kontroli zadań – prymitywny język poleceń dla monitora opisujący zadanie

- $JOB - nowe zadanie, określa czas działania w minutach
- $FORTRAN – załadowanie kompilatora języka Fortran z taśmy systemowej, bezpośrednio za nią następował program do skompilowania
- $LOAD - załaduj kod z taśmy
- $RUN - uruchom program
- $END - zakończ zadanie

- $IBSYS - System Supervisor jest wołany do pamięci rdzenia (core storage) i otrzymuje kontrolę
- $RELEASE – causes the unit assigned to the specified system unit function to be released from the function
- $IBEDT – System Supervisor woła System Editor do core storage z System Library Unit i przekazuje mu kontrolę
- $EXECUTE – precyzuje, który subsystem ma przeczytać i zinterpretować następne karty (po karcie execute)
- $STOP - określa koniec stosu zadań

Do jakich zastosowań używa się dziś systemów wsadowych?
Wiele dzisiejszych systemów wsadowych automatyzuje zadania, dzięki czemu interakcja z człowiekiem nie jest wymagana, chyba że coś pójdzie nie tak. Przykłady:

- payroll system: systemy wsadowe są idealne do tworzenia listy płac pracowników, wykonywanie obliczeń potrzebnych do wypłat pracownikom
- zarzadzanie ogromnymi bazami danych (np. w systemach rezerwacji miejsc lotniczych)
- w bankach transakcje o danej godzinie – są zbierane te wsady transakcji
- SLURM (dawniej *Simple Linux Utiliy for Resource Management*) to system kolejkowy (resource manager, job scheduler) działający np. na maszynach w ICM (centrum obliczeniowe UW) służących do obliczeń naukowych na komputerach dużej mocy

Źródła: slurm

L1.Z4. 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 systemu **interaktywnego**, który nie jest wieloprogramowy.

wieloprogramowe systemy wsadowe – rozwiązanie polegające na podzieleniu pamięci na kilka części i umieszczeniu w każdej z nich osobnego zadania. Podczas gdy jedno zadanie oczekiwało na zakończenie operacji IO, drugie mogło korzystać z procesora

systemy z podziałem czasu – odmiana systemów wieloprogramowych, w których każdy użytkownik posiadał podłączony do komputera terminal, który zapewniał użytkownikom interaktywną obsługę komputera. Podobnie jak system wieloprogramowy system ten może wykonywać wiele zadań jednocześnie. Istotną różnica między tymi systemami jest to, że system z podziałem czasu nie czeka na przestój jakiegoś programu, aby uruchomić inny, ale uruchamia wiele zadań jednocześnie. My - użytkownicy - mamy wrażenie, że zadania te wykonują się jednocześnie, ale jest to tylko nasze złudzenie. Tak naprawdę procesor przełącza się pomiędzy poszczególnymi zadaniami, ale przełączenia te następują tak szybko, że użytkownicy mogą na bieżąco współpracować z wszystkimi zadaniami w trybie rzeczywistym.

system interaktywny – pozwala wpływać użytkownikowi na SO podczas pracy innego programu

Jaka była motywacja do wprowadzenia wieloprogramowych systemów wsadowych?

- W komputerach 7094, w których bieżące zadanie było wstrzymywane do czasu zakończenia operacji z taśmą lub innej operacji IO, procesor główny pozostawał bezczynny do momentu zakończenia tej operacji.
- W przypadku obliczeń naukowych intensywnie wykorzystujących procesor operacje IO nie są częste, zatem nie powodowało to znaczącego marnotrawstwa czasu. W przypadku komercyjnego przetwarzania danych czas oczekiwania związany z IO wynosił często 80-90% całkowitego czasu, więc trzeba było coś z tym zrobić, by uniknąć tak wielkiego stopnia bezczynności drogiego procesora głównego.
- Pojawiło się rozwiązanie polegające na podzieleniu pamięci na kilka części i umieszczeniu w każdej z nich osobnego zadania. Podczas gdy jedno zadanie oczekiwało na zakończenie operacji IO, drugie mogło korzystać z procesora.
- Bezpieczne przechowywanie w pamięci wielu zadań naraz wymagało specjalnego sprzętu, który chroniłby każde z zadań przed „szpiegowaniem" oraz modyfikowaniem jednego zadania przez drugie, ale komputery z serii 360 oraz inne systemy z trzeciej generacji były wyposażone w taki sprzęt.

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

- Czas pomiędzy złożeniem zadania a otrzymaniem wyników często wynosił kilka godzin. Roszczeniowi programiści chcieli mieć szybciej te wyniki i możliwość debugowania programów.
- Z potrzeby szybkiej odpowiedzi powstała technika podziału czasu – odmiany systemów wieloprogramowych, w których każdy użytkownik posiadał podłączony do komputera terminal.
- Komputer zapewniał szybką, interaktywną obsługę wielu użytkownikom, a także pracę nad złożonymi zadaniami wsadowymi w tle w czasie, w którym w systemach bez podziału czasu procesor był bezczynny.
- Pierwszy komputer z podziałem czasu CTSS zbudowano w MIT.
- Generalnie idea: do przestojów na I/O dochodzą jeszcze przestoje na interakcję usera, który to robił tę interakcję przez terminal.

Podaj przykład systemu interaktywnego, który nie jest wieloprogramowy.

- MS-DOS.
- CP/M na Intel 8080.

L1.Z5. Bardzo ważną zasadą przy projektowaniu oprogramowania, w tym systemów operacyjnych, jest rozdzielenie **mechanizmu** od **polityki**. Wyjaśnij te pojęcia, odnosząc się do powszechnie występujących rozwiązań, np. otwieranie drzwi klasycznym kluczem versus kartą magnetyczną.

polityka – zestaw reguł, które system ma przestrzegać, np. polityka zamykania drzwi. Pytanie: jak zaaranżować elementy?
mechanizm – praktyczna realizacja polityki, np. instalowanie zamków w drzwiach i otwieranie ich kluczem. Pytanie: jak można użyć elementów?

Dla każdej polityki mogą istnieć różne mechanizmy, np. nie tylko kluczem można otworzyć drzwi – także kartą magnetyczną. Separacja mechanizmu od polityki – mechanizm nie powinien narzucać polityki.

Otwieranie drzwi kartą:
Polityka – karta z odpowiednim kodem może otworzyć drzwi.
Mechanizm - zdefiniowane kody otwierają drzwi.

- mechanizm w tym przykładzie nie narzuca polityki, nie definiuje, kto i kiedy ma wchodzić
- łatwa zmiana polityki (wprowadzenie nowych polityk – grupy, klasy dostępu), wymaga zmiany parametrów mechanizmu

Otwieranie drzwi kluczem:
Polityka – drzwi otwierane są przez pasujący klucz.
Mechanizm – drzwi posiadają zamek, który może być otwarty tylko przez pasujący klucz.

- jeśli chcesz zmienić osobę, która może otworzyć drzwi, musisz zmienić zamki i klucze - przy zmianie polityki trzeba zmienić też mechanizm