# 1. Wstęp

W obecnych czasach, prowadzenie symulacji przed wykonaniem jakiejś czynności w rzeczywistości stało się czymś całkowicie normalnym. Pozwala to dobrać optymalne parametry i/lub sposób wykonania danej rzeczy. By określić najlepsze parametry, często przeprowadza się kilka, kilkanaście lub nawet kilkaset i więcej symulacji tego samego procesu ze zmienionymi parametrami procesu. Przeprowadzenie każdej z tych symulacji, zajmuje pewną ilość czasu. Celem tej pracy jest sprawdzenie, czy dzięki zmianom parametrów lub algorytmów funkcji odpowiedzialnych za przydział zasobów systemu operacyjnego, możliwe jest zauważalne skrócenie czasu symulacji, przy zachowaniu poprawnych wyników.

W tej chwili na rynku, występuje wiele różnych systemów operacyjnych. Są to zarówno rozwiązania komercyjne, o zamkniętym kodzie źródłowym (jak rodzina systemów operacyjnych Microsoft Windows), jak i systemy o otwartym kodzie źródłowym, rozwijane głównie przez społeczność. Wśród tych ostatnich możemy wyróżnić rozbudowane systemy jak np. GNU/Linux czy FreeBSD, ale również mniejsze projekty, często tworzone przez pojedyncze osoby. Te mini-systemy operacyjne, pisane zazwyczaj od podstaw bardzo często nie są systemami ogólnego przeznaczenia (twórca nie zakłada że jego system będzie używany zarówno na laptopie do pracy biurowej i na serwerze do udostępniania plików). Ukierunkowanie rozwoju systemu na konkretne zastosowanie, pozwala na dobranie oraz dostrojenie optymalnych algorytmów, pozwalających uzyskanie maksymalnej wydajności oraz stabilności w danym zastosowaniu. Dzięki takiemu podejściu, możliwe jest maksymalne wykorzystanie posiadanych zasobów sprzętowych, a co za tym idzie skrócenie czasu potrzebnego do wykonania danej operacji.

W przypadku dużych systemów o otwartym kodzie źródłowym, które mają za założenie być uniwersalne (jak np. GNU/Linux), często w wypadku gdy jest zapotrzebowanie, pojawiają się odmiany, posiadające zmodyfikowany kod źródłowy, aby ułatwić oraz poprawić działanie systemu w konkretnym zastosowaniu. Przykładem dystrybucji systemu GNU/Linux, nastawionej na zastosowanie w instytucjach naukowych jest Scientific Linux.

Symulacje procesów metalurgicznych, w większości opierają się na obliczeniach numerycznych (np. metodą elementów skończonych), więc najbardziej obciążonymi podzespołami komputera są procesor, który zajmuje się wykonywaniem obliczeń matematycznych oraz pamięć operacyjna, w której przechowywane są dane, wyniki cząstkowe oraz końcowe wyniki obliczeń.

## 2. Budowa systemu operacyjnego

Systemem operacyjnym, nazywamy całość oprogramowania zarządzającego sprzętem komputerowym oraz tworzącego środowisko do uruchamiania i kontroli aplikacji użytkownika. Współczesny system operacyjny powinien realizować funkcje takie jak:

- planowanie przydziału czasu procesora,
- zarządzanie pamięcią operacyjną,
- kontrolę urządzeń podłączonych do komputera,
- synchronizację, kontrolę oraz współdzielenie dostępu do urządzeń oraz pamięci operacyjnej,
- kontrolę uprawnień użytkowników pracujących w systemie.

Współczesne systemy operacyjne, możemy podzielić na trzy podstawowe części:

- jądro systemu,
- sterowniki urządzeń,
- powłokę.

Jądro systemu operacyjnego (ang. *kernel*), jest to najważniejsza część całości oprogramowania, które nazywamy systemem. To właśnie jądro jest pierwszym elementem systemu, ładowanym przez program ładujący (ang. *bootloader*). Po załadowaniu, musi ono wykonać pełną inicjalizację podstawowego sprzętu (jak procesor czy pamięć operacyjna) oraz przygotować środowisko do pracy sterowników urządzeń i aplikacji użytkownika. Zarówno sterowniki jak i aplikacje, komunikują się między sobą oraz z jądrem za pomocą dostarczanego przez nie API (ang. *Application Programming Interface*). Współczesne jądra systemów, możemy podzielić na kilka typów, jednak najczęściej spotykane są trzy typy:

- jądro monolityczne – sterowniki urządzeń pracują w przestrzeni adresowej oraz na poziomie uprzywilejowania jądra, rozwiązanie to jest proste oraz wydajne. Jednak poprzez współdzielenie pamięci oraz poziomu uprzywilejowania z jądrem i innymi sterownikami, wadliwie napisany lub złośliwie spreparowany sterownik może spowodować niestabilną pracę całego systemu.
- Mikrojądro – sterowniki i większość usług systemu (jak np. system plików), są odseparowane od jądra i pracują w przestrzeni użytkownika. Pozwala to na ograniczenie dostępu dla sterowników, do niezbędnego minimum. Dzięki temu, to rozwiązanie jest bardziej bezpieczne oraz stabilniejsze. Wadą tego rozwiązania jest trudniejsza implementacja oraz spadek wydajności, wynikający z konieczności

przełączania kontekstu procesora oraz konieczności wykorzystania jądra do realizacji pewnych niskopoziomowych funkcji. Przykładem mikrojądra jest GNU Hurd

- jądro hybrydowe – łączy cechy obydwóch powyższych typów, część sterowników i usług pracuje w przestrzeni adresowej jądra a część działa w przestrzeni użytkownika. Przykładem systemu korzystającego z tego typu jądra jest Microsoft Windows (rodzina NT).

W jądrze nowoczesnego systemu operacyjnego możemy wyróżnić kilka części:

- kod inicjujący – kod wykonywany tylko raz, podczas uruchamiania jądra, w niektórych systemach (np. Linux), pamięć zajmowana przez ten kod, po załadowaniu systemu jest zwalniana,
- manager urządzeń – odpowiedzialny za komunikację ze sterownikami oraz zarządzanie podłączonym sprzętem,
- manager pamięci – odpowiedzialny za zarządzanie pamięcią fizyczną, pamięcią jądra systemu, plikami wymiany oraz pamięcią wirtualną aplikacji
- planista – odpowiedzialny za przydzielanie czasu procesora dla poszczególnych procesów i wątków,
- system plików – odpowiedzialny za szeroko pojęte operacje na plikach, katalogach oraz systemach plików (otwierania, zapisywanie, kasowanie, itp.)
- synchronizacja oraz komunikacja między procesami – moduł, odpowiedzialny za wymianę komunikatów pomiędzy procesami czy wątkami oraz synchronizację i kontrolę dostępu do danych jądra, pamięci współdzielonej, itp.

Sterowniki urządzeń są to moduły, które do urządzeń danego typu (np. karta graficzna), tworzą w systemie usystematyzowany (w danym systemie) interfejs, z którego korzysta jądro oraz aplikacje użytkownika. Dzięki wprowadzeniu takiego interfejsu, nie jest konieczna wiedza o konkretnym modelu urządzenia z którego korzystamy, a jedynie o jego typie (np. karta sieciowa). Stworzenie takiego interfejsu oraz separacje sterowników od jądra, czyni system operacyjny przenośny między różnymi konfiguracjami sprzętowymi.

Powłoka (ang. *shell*) jest to aplikacja, której zadaniem jest umożliwienie komunikacji użytkownika z jądrem systemu. Dostarcza ona podstawowe narzędzie do pracy w systemie oraz uruchamiania innych aplikacji. Istnieją dwa główne typy powłok:

- tekstowe (ang. *CLI - Command line intereface)* – praca z tego typu powłoką, polega na wprowadzaniu poleceń tekstowych, które są następnie interpretowane i na ich podstawie wykonywane są konkretne akcje (np. uruchomienie aplikacji). Przykładem

powłoki tekstowej jest GNU Bash

- graficzne (ang. *GUI – Graphical user interface*) – ten typ powłoki obsługiwany jest głównie za pomocą urządzenia wskazującego (np. myszy czy touchpada), a korzystanie z niego polega na wybieraniu odpowiednich akcji, poprzez wskazywanie oraz klikanie na interaktywnych elementach powłoki, nazywanych kontrolkami. Przykładem tego typu powłoki, jest explorer z systemu Microsoft Windows.

## 3. Wykorzystane systemy operacyjne

Przy pisaniu niniejszej pracy, zostały, wykorzystane dwa systemy operacyjne.

### 3.1 IdyllaOS

IdyllaOS jest to prosty system operacyjny, dostępny na licencji GNU GPL w wersji 3. System tworzony jest hobbystycznie, nie nadaje się do codziennego użytku. Jednak dzięki prostocie, podstawowe funkcje oraz algorytmy można łatwo modyfikować. Całość kodu systemu dostępna jest na repozytorium GIT. System jest zgodny ze standardem POSIX, co umożliwia bardzo proste przenoszenie na niego aplikacji, z innych systemów zgodnych z tym standardem. Podstawą systemu jest jądro monolityczne z możliwością dynamicznego ładowania modułów, które są relokowane i linkowane z jądrem. Dodatkowym założeniem jest, to że jądro nie zawiera w sobie żadnych sterowników, poza tymi które są niezbędne (jak np. sterownik karty graficznej, pracujący w trybie tekstowym VGA). Dodatkowo, kod jądra podzielony jest na dwie części:

- część zależną od architektury sprzętowej – w tym kodzie, znajdują się niskopoziomowe funkcje, operujące bezpośrednio na sprzęcie (jak np. funkcja do zmiany kontekstu procesora),
- część nie zależną od architektury sprzętowej – w tej części znajdują się wszystkie funkcje, które mogą być wykorzystywane na wielu architekturach sprzętowych (np. manager pamięci czy obsługa wirtualnego systemu plików)

Taki podział kodu, umożliwia łatwe przenoszenie systemu pomiędzy architekturami sprzętowymi. W chwili obecnej, system posiada wsparcie dla 32-bitowej architektury Intel IA-32 oraz 64-bitowej architektury AMD64 (przy czym, wersja 64-bitowa nie posiada zaimplementowanych wszystkich funkcji, przez co posiada ograniczoną funkcjonalność).