1. Omówić podział podział architektur komputerów według TAKSONOMI FLYNNA

- SISM Single instruction single data stream
- SIMD Single instruction multiple data stream
- MISD Multiple instructions single data stream (rzadko spotykana)
- MIMD Multiple instructions multiple data stream (najczęśniej spotykana)

W skrócie, SISD to pojedynczy procesor wykonujący jedną instrukcję na jednym zestawie danych, SIMD to wiele procesorów wykonujących tę samą instrukcję na wielu zestawach danych, MISD to rzadka architektura z wieloma instrukcjami operującymi na jednym zestawie danych, a MIMD to najbardziej powszechna architektura z wieloma procesorami i instrukcjami działającymi na wielu zestawach danych jednocześnie.

2. Omówić pojęcia - program sekwencyjny, równoległy oraz współbieżny

- Program sekwencyjny to taki który wykonuje się krok po kroku i każda operacja oczekuje na poprzednią we wcześniej ustalonej kolejności
- Program równoległy to taki w którym wiele instrukcji wykonuje się jednocześnie i zazwyczaj są od siebie kompletnie niezależne
- program współbieżny to taki w którym wiele instrukcji wykonuje się jednocześnie i zazwyczaj są ze sobą w jakiś sposób powiązane

3. Omówić różnice pomiędzy procesem a wątkiem

- Proces jest większą jednostką organizacyjną która posiada swój własny adres w pamięci
- Procesy są od siebie odizolowane, w odróżnieniu od wątków które współdzielą ze sobą pamięć w zakresie danego procesu
- Synchronizacja między procesami jest dość utrudniona przez stopień izolacji i wymaga dodatkowych narzędzi, gdzie wątki z racji że dzielą ze sobą pamięć i zasoby można ze sobą dość łatwo zsynchronizować
- Błąd w procesie nie wpływa na inne procesy, za to błąd w wątku wpływa znacznie na proces w zakresie którego pracuje.

4. Omówić poprawność programów równoległych (do omówienia zakleszczenie, poprawność częściowa, zagłodzenie)

- Deadlock(zakleszczenie) Występuje w sytuacji gdy więcej niż jeden proces, zostaje zablokowany z dowolnego powodu, co prowadzi do tego że pozostałe procesy także nie są w stanie pracować. Aby tego uniknąć używamy mechanizmów synchronizacji takich jak np. semafory czy mutexy

- Poprawność częściowa Polega to na tym że program wykonuje się, jednak nie do końca w sposób w jaki powinien działać Zazwyczaj jest to dopuszczalne ale nie zalecane, wraz z upływem czasu i rozwojem programu może doprowadzić do następnych problemów.

- Starvation(zagłodzenie) Problem w którym proces lub wątek nie dostaje odpowiedniej ilości zasobów lub czasu procesora co może doprowadzić do utraty synchronizacji i problemów z działaniem programu. Zazwyczaj możemy uniknąć tego dzięki mechanizmom kontroli zasobami lub synchronizacji.

5. Opisać problem wzajemnego wykluczenia

Występuje on gdy kilka procesów lub wątków próbuje korzystać z tych samych zasobów przez co blokują się wzajemnie, co może skutkować wyścigiem o zasoby i może doprowadzić do niechcianego rezultaty w programie, prostym rozwiązaniem są zazwyczaj mechanizmy synchronizacji takiego jak semafory, bloki krytyczne i zmienne warunkowe.

6. Wyjaśnić co to jest sekcja krytyczna i po co ją stosujemy

Sekcja krytyczna to część kodu w zakresie której tylko jeden wątek/proces może korzystać z zasobów w tym samym czasie. Prostą metodą implementacji jest użycie mechanizmów synchronizacji aby ograniczyć działające w tym samym czasie procesy. Głównym powodem dlaczego powinno się z nich korzystać jest to że zapobiegają wyścigowi o zasoby.

7. Opisać problem producentów-konsumentów

Jest to przykład działania programu współbieżnego z wykorzystaniem mechanizmów synchronizacji. Problem polega na tym że jest produkowany produkt, który następnie jest wykorzystywany przez konsumentów gdzie oba wątki korzystają z tego samego magazynu lub bufora. Źle rozwiązany problem zwraca nam 0 lub narastająco w nieskończoność wynik ponieważ jest podstawowym przykładem wyścigu o zasoby.

8. Opisać problem czytelników i pisarzy

Jest to problem synchronizacji polegający na tym że mamy wspólną bibliotekę oraz pisarzy i czytelników gdzie pisarze zapisują do biblioteki a czytelnicy z niej odczytują, musimy pracować ze współdzielonymi zasobami w taki sposób żeby nie doszło do zagłodzenia żadnego z użytkowników biblioteki.

9. Opisać problem ucztujących filozofów

Kolejny z problemów synchronizacji tym razem polegający na tym że mamy jeden wspólny zasób wykorzystywany do jedzenia "pałeczki" i 5 filozofów którzy stopniowo stają się głodni więc muszą skorzystać z pałeczek. Główne trudności w tym problemie to unikanie zagłodzenia i zakleszczenia

10. Opisać modele współbieżności (scentralizowane, rozproszone)

- Model wpółbieżności scentralizowanej

Wszystkie decyzje w tym systemie dzieją się z jednego scentralizowanego punktu, z niego jest podejmowana decyzja o tym jakie procesy mają dostać dostęp do zasobów. Jest to zdecydowanie prostsze do implementacji rozwiązanie, łatwe do zrozumienia i proste w utrzymaniu.

- Model współbieżności rozproszony

Podejmowane przez system decyzje są rozproszone i każda z nich jest podejmowana lokalnie. Rozwiązanie to jest trudniejsze do zaimplementowania jednak jest bardziej odporny na awarię i cechuje się większą skalowalnością.

11. Opisać różnice pomiędzy komunikacją synchroniczną a komunikacją asynchroniczną

- Komunikacja synchroniczna opiera się na oczekiwaniu na siebie poszczególnych elementów programu, gdzie w asynchronicznej wszystko dzieje się mniej więcej równocześnie
- Zazwyczaj synchroniczna stosowana jest w prostych problemach ze względu na to że jest dość podatna na zakleszczenie, więc tym bardziej skomplikowany program tym większa szansa że coś takiego może się wydarzyć, jednak jest ona nieco prostsza w implementacji i w zrozumieniu

- Główną przewagą komunikacji asynchronicznej jest to że jeśli dobrze ją skonstruujemy to może znacząco wpłynąć na szybkość wykonywania operacji, a także możemy dzięki niej zabezpieczyć program przed niechcianymi zakleszczeniami. Niestety jest przy tym nieco trudniejsza do zrozumienia i nieco bardziej złożona.

12. Co to jest semafor - wymienić rodzaje semaforów

Semafory w unixowym języku C są strukurą służącą do synchronizacji procesów, w zależności od typu semaforu ma nieco inne zastosowanie i działa nieco różnie

- Semafor binarny Podstawowym semaforem jest semafor binarny który przyjmuje tylko wartości binarne, możemy za jego pomocą synchronizować procesy które wymagają tylko dwóch stanów, zazwyczaj za ich pomocą wstrzymujemy program aż do oczekiwanego etapu.
- Semafor ogólny (zliczający) Działa podobnie do binarnego, jednak przyjmuje większy zakres liczb a dokładniej wszystkie nieujemne liczby całkowite co znacząco może wpłynąć na swobodę i możliwości zarządzania synchronizacją programu.
- Semafor uogólniony Działa tak samo jak ogólny z taką różnicą że jeśli semafor ten przyjmie wartość ujemną to zostanie zablokowany
- Semafor dwustronnie ograniczony Tak samo jak w przypadku ogólnego z różnicą taką że posiada także górne ograniczenie ustalone przy inicjacji semaforu

13. Zapisać definicję ogólną oraz praktyczną semafora ogólnego

Zmienna całkowita, niemniejsza od zera z dostępnymi dwoma operacjami

- Opuszczenia (wait | P) if S > 0 then S -= 1
- Podniesienia (signal | V) s += 1

Wait

```
if (S > 0)
    S = S - 1;
else
    wstrzymaj
```

Signal

```
if (cokolwiek)
    dzialaj;
else
    S = S + 1
```

14. Zapisać definicję ogólną oraz praktyczną semafora binarnego

Zmienna binarna z dostępnymi dwoma operacjami

- Opuszczenia (wait | P) if S = true then S = false
- Podniesienia (signal | V) s = true

Wait

```
if (S == true)
    S = false;
else
    wstrzymaj
```

Signal

```
if (cokolwiek)
    dzialaj;
else
    S = true
```

15. Zapisać definicję ogólną oraz praktyczną semafora dwustronnie ograniczonego

Semafor jest zmienną całkowicie liczbową niemniejszą od zera która zawiera się w przedziale [0, N]

Wait

```
if (S == 0)
    wstrzymaj;
else if (cokolwiek)
    działaj;
else
    S = S - 1
```

Signal

```
if (S == N)
    wstrzymaj;
else if (cokolwiek)
    działaj
else
    S = S + 1
```

16. Zapisać problem wzajemnego wykluczenia z użyciem semaforów

```
const int N = 5;
BinarySemaphore S = true;
```

```
void P(int procid)
{
    while (true)
    {
        funkcja();
        PB(S); //Opuszczenie semafora
        sekcja_krytyczna();
        VB(S); //Podniesienie semafora
    }
}
```

17. Zapisać problem producentów konsumentów z użyciem semaforów

```
const int N = 100; //Rozmiar buffora
Semaphore empty = N;
Semaphore full = 0;
Produkt bufor[N];

void Producent()
{
    Produkt p;
    int j = 1;
    while (true)
    {
        produkuj(&p);
        P(empty);
        bufor[j-1] = p;
        j = j % N + 1;
        V(full)
    }


void Konsument()
{
    produkt P;
    int k = 1;
    while (true)
    {
        P(full);
        p = bufor[k-1];
        K = k % N + 1;
        V(empty);
        konsumuj(p);
    }
}
}
```

18. Zapisać problem czytelników i pisarzy z użyciem semaforów

```

```

```
const int R = 5; //Czytelnicy
const int W = 2; //Pisarze

Semaphore empty = R;
BinarySemaphore W = true;

void Czytelnik(int idCzytelnika)
{
    while (true)
    {
        something();
        P(empty);
        czytanie();
        V(empty);
    }
}

void Pisarz(int idPisarza)
{
    while (true)
    {
        something();
        PB(W);
        for (int i = 1; i <= R; i++)
        {
            P(empty);
        }
        pisanie();
        for (int i = 1; i <= R; i++)
        {
            V(empty);
        }
        VB(W);
    }
}
```

19. Zapisać problem ucztujących filozofów z użyciem semaforów

```
BinarySemaphore paleczki[] = {true, true, true, true, true};
const int filozofowie = 5;
Semaphore lokaj = 4;

void Filozof(idFilozofa)
{
    while (true)
    {
        for (int i = 1; i <= filozofowie; i++)
        {
            think();
            P(lokaj);
            PB(paleczki[i]);
```

```
            PB(paleczki[(i+1)%5]);
            jedzenie;
            VB(paleczki[i]);
            VB(paleczki[(i+1)%5]);
            V(lokaj);
        }

    }
}
```

20. Czym są monitory w programowaniu współbieżnym i po co je stosujemy

Monitor stanowi połączenie modułu programistycznego z częścią krytyczną, zawiera on deklaracje stałych oraz zmiennych funkcji, jednak są one lokalne tylko w monitorze.

```
class Monitor
{
    private:
        definicje;
        deklaracje
    public:
        Monitor()
        {
            instrukcje inicjujące
        }
}
```

Zazwyczaj monitory wykorzystujemy w zmiennych warunkowych z blokowaniem lub bez blokowania

21. Co to jest zmienna warunkowa i jak działa

zmienna warunkowa jest mechanizmem synchronizacji, wykorzystywanym zazwyczaj wraz z monitorami, umożliwia wątkom czekanie na spełnienie pewnego warunku, a także sygnalizowanie innym wątkom o istnieniu tego warunku.

- Oznaczamy je symbolem C (condition)
- Umożliwia wstrzymywanie/wznawianie procesów
- Posiada własną kolejkę

22. Zapisać problem producentów konsumentów z użyciem zmiennej warunkowej

```
class Bufor: Monitor
{
    private:
        const int N = liczbaProduktow
        Porcja bufor[N];
        int ile, doWstawienia, doWyjecia;
        condition producenci, konsumenci;
    public:
```

```
        void Wstaw(Porcja element)
        {
            if (ile == N)
            {
                wait(producenci);
            }
            else
            {
                doWstawienia = (doWstawienia + 1) % N;
                bufor[doWstawienia] = element;
                ile++
                signal(konsumenci);
            }
        }

        void Pobierz(Porcja& element)
        {
            if (ile == 0)
            {
                wait(konsumenci);
            }
            else
            {
                doWyjecia = (doWyjecia + 1) % N;
                bufor[doWyjecia] = element;
                ile--
                signal(producenci);
            }
        }
}
```

23. Zapisać problem ucztujących filozofów z użyciem zmiennej warunkowej

```
class Czytelnia: Monitor
{
    private:
        int czyta, pisze;
        condition czytelnicy, pisarze;
    public:
        void StartCzytanie()
        {
            if (pisze > 0)
            {
                wait(czytelnicy);
            }
            else
            {
                czyta++;
            }
        }
        void StopCzytanie()
        {
```

```
            czyta--
            if (czyta == 0)
            {
                wait(pisarze)
            }
        }
        void StartPisanie()
        {
            if (czyta + pisze > 0)
            {
                wait(pisarze);
            }
            else
            {
                pisze = 1;
            }
        }
        void StopPisanie()
        {
            pisze = 0;
            if (empty(czytelnicy))
            {
                signal(pisarze);
            }
            else
            {
                do
                {
                    signal(czytelnicy)
                } while (!empty(czytelnicy))
            }
        }

        Czytelnia(){
            pisze = 0;
            czyta = 0;
        }
}
```

24. Zapisać problem czytelników -pisarzy z użyciem zmiennej warunkowej

```
class Widelce: Monitor
{
    private:
        int wolne[5];
        condition filozof[5];
    public:
        void Biore(int i)
        {
            while (wolne[i] < 2)
            {
                wait(filozof[i]);
```

```
            }
            wolne[(i+4)%5] = wolne[(i+4)%5] - 1;
            wolne[(i+1)%5] = wolne[(i+1)%5] - 1;
        }

        void Odkladam()
        {
            wolne[(i+4)%5] = wolne[(i+4)%5] + 1;
            wolne[(i+1)%5] = wolne[(i+1)%5] + 1;
            signal(filozof[i+4]%5);
            signal(filozof[i+1]%5);
        }
        Widelce()
        {
            for (int i = 0; i <5; i++)
            {
                wolne[i] = 2;
            }
        }
}
```

25. Omówić monitor Hoara oraz Massy

Oba te monitory są mechanizmami synchronizacji które wykorzystają zmienne warunkowe, jednak są nieco różne w tym jak obsługują warunki i sygnalizacje.

- Monitor Hoara obsługuje warunki za pomocą których wątki mogą oczekiwać na spełnienie konkretnego warunku
- Monitor Hoara jest bardziej ogólny i wszechstronny, jednak trzeba z nim uważać ponieważ nieprawidłowe użycie może prowadzić do zakleszczeń.
- Monitor Massy korzysta jednocześnie z dwóch rodzajów warunków: warunku zmiennego i warunku zdarzenia
- Monitor Massy wprowadza dodatkową warstwę sygnalizacji przez co jest bardziej bezpeiczny

26. Omówić etapy projektowania algorytmu równoległego

    1. Dekompozycja problemu obliczeniowego na zadania
    2. Analiza rozdrobnienia obliczeń
    3. Minimalizacja kosztu algorytmu równoległego
    4. Przydzielenie zadań procesom

27. Omówić metody dekompozycji stosowane w przetwarzaniu współbieżnym

    1. Dekompozycja danych
    2. Dekompozycja funkcjonalna
    3. Stopień współbieżności problemu Liczba zadań które mogą być rozwiązywane równocześnie

28. Opisz Prawo Amdahla oraz wnioski z niego wynikające

Prawo Amdahla opisuje jakie maksymalne przyśpieszenie możemy osiągnąć poprzez równoległe przetwarzanie.

Nie działa to jednak tak prosto ponieważ konieczna jest równomierność rozłożenia pracy, jeśli któryś z wątków będzie przeciążony to możemy otrzymać odwrotny rezultat.

Trzeba też pamiętać że zazwyczaj stosuje się je dla dużych zbiorów danych lub zadań które można bardzo mocno rozłożyć.

29. Wyjaśnij pojęcia przyspieszenia oraz wydajności algorytmu równoległego

Przyśpieszenie algorytmu równoległego jest miarą ilustrującom, ile razy algorytm zrównolegniony jest szybszy od swojej wersji sekwencyjnej działając na jednym procesorze.

30. Wyjaśnij kiedy możemy mieć do czynienia z przyspieszeniem superliniowym

Jest to niezwykle rzadka sytuacja w której przyśpieszenie algorytmu równoległego jest większe niż liczba użytych w nim procesów.

Może on mieć miejsce w takich sytuacjach jak:

- Gdy zadanie wymaga intenstywnej ilości pamięci, a dostęp jest czasochłonny może dojść do wyżej opisanej sytuacji.
- Jeśli algorytm równoległy korzysta w pełni z pamięci współdzielonej.
- Optymalizacja w rzadkich przypadkach doskonała optymalizacja pracy programu może doprowadzić do takiego zdarzenia.

31. Wymień źródła narzutów przetwarzania równoległego

- Narzut komunikacyjny
- Narzut synchronizacji
- Narzut zarządzania zasobami
- Narzut tworzenia i zakańczania wątków

32. Jak obliczamy koszt algorytmu równoległego oraz całkowity narzut równoległy

Koszt algorytmu równoległego to suma kosztów czasowych zadań wykonywanych równolegle.

Całkowity narzut równoległy to dodatkowy czas spędzony na komunikację, synchronizację i inne operacje związane z przetwarzaniem równoległy