1. **Co to jest zdarzenie w standardzie POSIX i jakie możliwości obsługi zdarzenia przewiduje standard.**

Zdarzenie to rozszerzenie pojęcia sygnału. Jest to mechanizm do powiadamiania procesów przez system operacyjny o zdarzeniach np. błędach (nienormalnych zdarzeniach które wydarzyły się podczas wykonywania procesu). W takiej sytuacji system operacyjny dostarcza sygnał, który przekazuje informację o tym, co się zdarzyło i sygnalizuje potrzebę jego obsłużenia. Pomysł zdarzenia jest taki, ze powiadomienie o nim, nie jest jedyną możliwoscią. W przypadku nienormalnej sytuacji proces1 mógłby zrobić coś więcej niż tylko powiadomić o tym proces2. Lub np. jeśli dopuścimy wielowątkowość procesów, możemy odpalić nowy wątek i przekazać mu obsługę tego zdarzenia.

Mechanizm zdarzeń tj. supersygnałów, których obsługa może być złożona. Może polegać na dostarczeniu tradycyjnego sygnału, ale także odpaleniu wątku w procesie. Mamy 3 rodzaje sytuacji:

- nic nie zrobimy

- wyślemy sygnał

- przekażemy obsługę zdarzenia do nowego wątku.

Zdarzenie opisuje struktura sigevent (wykład 8)

1. Wyjaśnić cel, mechanizm działania i ograniczenia protokołu dziedziczenia priorytetu.
2. Napisać szkic programu, który dokładnie co sekundę próbuje odczytać wiadomości z dwóch różnych kolejek.
3. Usługi warstwy 2 Profibus
4. **Szkielet programu, który tworzy trzy procesy potomne (zapisane w plikach pp1, pp2, pp3) i oczekuje na zakończenie pracy jednego z nich.**

*pid fork()* – tworzy nowy proces i tablicę stanu – identyczną kopię procesu macierzystego – dziedziczy przestrzeń adresową, wskaźniki, otwarte pliki i semafory

*pid* – id procesu potomnego || 0 gdy jesteśmy w procesie potomnym

*int exec() –* zmiana obszaru danych i obszaru programu

*posix\_spawn()* – tworyz nowz proces, dla którego definiujemy argumentami dziedziczenie

int status;

pid = fork();

if(pid !=0) { … }

else {

exec(pp1,…);

}

pid = fork();

if(pid !=0) { … }

else {

exec(pp2,…);

}

pid = fork();

if(pid !=0) { }

else {

exec(pp3,…);

}

wait(&status);

1. **Szkielet programu, który tworzy trzy procesy potomne (zapisane w plikach pp1, pp2, pp3) i oczekuje na zakończenie pracy tego drugiego**

Int status;

Pid1 = fork();

If(pid!=0) { … }

Else {

Exec(pp1,…);

}

Pid2 = fork();

If(pid!=0) { …}

Else {

Exec(pp2,…);

}

Wait\_pid(pid2, &status);

1. **Sposób adresowania wiadomości w sieci CAN.**

Poszczególne węzły nie mają adresu, jest to sieć magistralowa o topologii rozgłoszeniowej. Nie ma adresacji węzłów, są numerowane (adresowane) zmienne. Każda zmienna w tej sieci ma unikalne ID. Sieć jest bezpołączeniowa, wszystkie węzły rozgłaszają wiadomości broadcastem do innych węzłów. Po ID jest rozpoznawane czy nas to interesuje czy nie.

1. **Szkielet programu monitora realizującego stos liczb całkowitych w systemie zgodnym ze standardem POSIX.**

Int buf[MAX];

Pthread\_mutex\_t m;

Pthread\_cond\_t c;

Push(int value) {

Mutex\_lock(m);

Poloz(buf, value);

++s;

If(s == 1) {

Cond\_signal(c);

Mutex\_unlock(m);

}

}

Pop(int value) {

Mutex\_lock(m);

While( s==0) {

Cond\_wait(c,m);

Zdejmij(buf, value);

--s;

Mutex\_unlock(m);

}

1. **Szkielet programu procesu, który tworzy semafor nienazwany i wykorzystuje go do synchronizacji dostępu do zasobu dzielonego ze swoim procesem potomnym.**

Sem\_t sem;

Int shared = 1;

Int value = 0;

File \*fd; // deskr.pliku

Fd = shm\_open(“semafor”,…(prawa dostępu);

Ftruncate(fd, sizeof(sem\_t)); //alokacja pamięci

Addr = mmap(0, sizeof(sem\_t), flagi, fd, offset); // wskaźnik na wspólny obszar

Sem = (sem\_t\*)addr;

Sem\_init(sem, 1 ,0 );

// inny process

Fork();

Sem\_wait(sem);

Set\_channel(a); }

X = read() } – sekcja krytyczna

… }

Sem\_post(sem);

Sem\_destroy(sem);

Munmap(addr, sizeof(sem\_t)); // przeciwieństwo mmap

Close(fd);

Shm\_unlink(„semafor”);

1. Zdania charakteryzujące sieci przemysłowe
   * Muszą efektywnie przenosić krótkie wiadomości
   * Muszą efektywnie przenosić długie wiadomości
   * Sieć CAN wdraża priorytetowe szeregowanie wszystkich wiadomości nadawanych w sieci
   * Sieć Interbus jest siecią typu master-slave z wieloma węzłami typu master

* Wykonanie pomiaru i odczytanie przez węzeł nadrzędny temperatury z szybkiego czujnika dołączonego do węzła podporządkowanego wymaga jednego cyklu odpytania
* Sieć Interbus gwarantuje przekazanie wartości wszystkich zmiennych procesowych w pewnym z góry znanym czasie
* Węzły typu slave sieci Profibus nie mogą wywoływać żadnych usług warstwy liniowej
* W warstwie liniowej sieć profibus używa algorytmu dostępu z odpytywaniem
* W warstwie liniowej sieć profibus używa algorytmu dostępu znacznikowego
* Algorytm dostępu z przekazywaniem znazcnika gwarantuje deterministyczny czas nadania wszystkich komunikatów
* Długość okresu, przez który węzeł typu *master* sieci Profibus może nadawać swoje komunikaty jest ograniczona od góry przez wartość ustaloną podczas konfiguracji sieci.
* Długość okresu, przez który węzeł typu *master* sieci Profibus może nadawać swoje komunikaty jest stałą ustaloną podczas konfiguracji sieci.
* Każde wywołanie usługi Read w relacji **MSAC** sieci Profibus powoduje wysłanie żądania do węzła serwera i odebranie jego odpowiedzi
* Każde wywołanie usługi Read w relacji **MSCY** sieci Profibus powoduje wysłanie żądania do węzła serwera i odebranie jego odpowiedzi
* Każde wywołanie usługi Write w relacji **MSAC** sieci Profibus powoduje wysłanie danych do węzła serwera i odebranie jego potwierdzenia.
* Warstwa aplikacyjna sieci Profibus realizuje bezpołączeniowy protokół komunikacyjny
* Warstwa aplikacyjna sieci Profibus realizuje model komunikacji klient-serwer, w którym rolę serwera pełni węzeł typu **master**.
* Warstwa aplikacyjna sieci Profibus realizuje model komunikacji klient-serwer, w którym rolę serwera pełni węzeł typu **slave**.

1. **Napisać szkielet programu cyklicznego o dokładnie określonym czasie cyklu**

Int timer\_create(struct sigevent \*evp, timer\_t\* timer)

Int timer\_delete(timer\_t\* timer)

Int timer\_settime(timer\_t \*timer, struct itimerspec \*value)

Int timer\_getoverrun(timer\_t \*timer)

Timer\_t licznik;

Struct itimerspec menspec;

Struct timespec interval, value;

Value.tv\_sec = value.tv\_nsec = 0;

Interval.tv\_sec = 1;

Interval.tv\_nsec = 0;

Menspec.it\_interval = interval;

Menspec.it\_value = value;

Timer\_create(&event,&licznik);

Timer\_settime(licznik, &menspec);

While() {

//cos…

Pause();

}

1. **Zaprojektować drajwer wielokanałowego przetwornika a/c w postaci biblioteki funkcji działającego w systemie zgodnym ze standardem POSIX. Drajwer nie powinien dopuszczać do inwersji priorytetów. (W9)**
   * **Lista funkcji**
   * **Narysować szkielety programów**

Funkcje:

Ac\_open();  
Ac\_read(sem,char);  
Ac\_close(sem); **Wykład 9.**

1. **Wyjaśnić cel, mechanizm działania i ograniczenia protokołu pułapu priorytetu.**Protokół pułapu priorytetu służy do minimalizacji (lub wręcz unikniecia) zjawiska inwersji priorytetów oraz zakleszczeń.

Wszystkie procesy, które ubiegają się o dany zasów otrzymują maxymalny priorytet przypisany do danego zasobu wraz z pułapem priorytetu. Jego wartość jest określana przez najwyższy priorytet z pośród zadań, które mogą o niego konkurować. Protokół jest przejrzysty i przewidywalny, posiada jednak pewne ograniczenia. Jeżeli dany proces ma długą sekcję krytyczną, może znacznie opóźnić wykonywanie procesów konkurujących o zasób, co z kolei może negatywnie wpłynąć na ograniczenia czasowe systemu. Należy to wziąć pod uwagę podczas projektowania.

1. **Wyjaśnić działanie protokołu z odpytywaniem.**

Jego cechą jest oddzielenie rytmu przekazywania od rytmu komunikacji. Master odpytuje slave, a on mu natychmiast odpowiada. Jak policzy, to slave składa odpowiedź do bufora wyjściowego i wysyła dopiero jak master go zapyta. Nie ma niebezpieczeństwa kolizji, bo tylko master wysyła żądania. Slave nic nie robi sam z siebie.

1. **Wyjaśnić działanie protokołu z przekazywaniem znacznika. (W10)**

Protokół ten jest niesprzeczny z protokołem odpytywania. Nie może jednak przekazywać długich wiadomości. Jest w nim TOKEN (znacznik), który wędruje po kolejnych węzłach sieci i idzie w ustalonym porządku. Jak ktoś ma znacznik to może wykonywać operacje, a pozostali czekają, dzięki czemu unikamy kolizji. Węzły równoprawne, występuje jednak narzut związany z przekazywaniem znacznika. Do wad należy możliwość utraty znacznika lub jego rozmnożenie.

1. **Do czego służą funkcje mlock i mlockall standardu POSIX i w jakich okolicznościach ich użycie jest konieczne? (wykład 8)**

*Proces rezydentny… ?*

Mlock() – dla fragmentu kodu

Wymiatanie – swapping, przesyłanie procesu pomiędzy pamięcią operacyjną i dyskową

Int mlockall(int flags) – wyłączenie pamięci wirtualnej w stosunku do całej pamięci adresowej procesu.

Flagi- FUTURE/CURRENT

Wadą jest to, że wdraża niepewność czy ten fragment jest w pamięci. Dostępność funkcji jest niedeterministyczna czasowo, ponieważ np. przerwanie musi być wykonane od razu.