Imię i nazwisko: _____ Numer indeksu: _____

Systemy operacyjne

5 lutego 2020

czas trwania: 180 minut

Punkty	0 – 49	50 – 59	60 – 69	70 – 79	80 – 89	90 – 100
Ocena	2.0	3.0	3.5	4.0	4.5	5.0

Zadania testowe: Zdania prawdziwe oznacz literą T, a fałszywe literą N. Funkcja p, która przyporządkowuje punkty liczbie poprawnie udzielonych odpowiedzi, jest zadana następująco: $p(4) = 3$, $p(3) = 2$, $p(2) = 1$, $p(1) = 0$, $p(0) = 0$.

Zadanie 1 (3). Fakty dotyczące szeregowania procesów w systemach uniksowych.

- T Procesy są wywłaszczane w wyniku przerwania zegarowego lub wybudzenia procesu o wyższym priorytecie.
- N Kwant czasu przydzielany przez algorytm szeregujący wynosi z reguły około jednej sekundy.
- N Planista zajmuje się nastawianiem zegara i przełączaniem procesów.
- T Przełączanie procesów wymaga zmiany kontekstu procesora i zmiany przestrzeni adresowej.

Zadanie 2 (3). Fakty dotyczące hierarchii procesów.

- N Proces może czekać na zakończenie swojego wnuka pod warunkiem, że zna jego `PID`.
- N Po wywołaniu «`execve`» proces nie może już zmienić swojej grupy procesów.
- N Procesy zombie są zawsze grzebane przez proces żniwiarza lub «`init`».
- T Sieroty to procesy, które straciły rodzica i zostały automatycznie przygarnięte przez żniwiarza lub «`init`».

Zadanie 3 (3). Dla których z poniższych sygnałów można ustalić procedurę obsługi?

- T `SIGSEGV`
- T `SIGTERM`
- N `SIGSTOP`
- N `SIGKILL`

Zadanie 4 (3). Fakty dotyczące wysyłania i odbierania sygnałów.

- N Zablokowanie danego sygnału dla procesu powoduje, że system ignoruje fakt wysłania sygnału do tego procesu.
- T Sygnał może zostać odebrany tylko przed powrotem do przestrzeni użytkownika.
- T Zignorowanie sygnału `SIGCHLD` powoduje, że dzieci są automatycznie grzebane.
- N Proces zatrzymany sygnałem `SIGSTOP` zostanie wznowiony po odebraniu dowolnego innego sygnału.

Zadanie 5 (3). Fakty na temat procedur obsługi sygnałów.

- T Proces ustawił pustą procedurę obsługi sygnału `SIGINT` przy pomocy «`signal`». Jeśli wyślemy do niego `SIGINT` dwa razy, to może się zdarzyć, że procedura obsługi zostanie wykonana tylko raz.
- N W procedurach obsługi sygnałów należy używać wyłącznie procedur wątkowo-bezpiecznych.
- T Sygnał s jest zablokowany na czas działania procedury obsługi sygnału s.
- T Wywołanie «`sigprocmask`» może posłużyć do realizacji sekcji krytycznej chroniącej przed wyścigami na zmiennych wykorzystywanych w procedurze obsługi sygnałów.

Zadanie 6 (3). Które z poniższych zasobów przeżywają wywołanie systemowe `execve`?

- [N] procedury obsługi sygnałów
- [T] tablica deskryptorów plików
- [N] wątki
- [N] współdzielona pamięć anonimowa

Zadanie 7 (3). Które z poniższych zasobów są dziedziczone przez proces utworzony wywołaniem `fork`?

- [T] identyfikator grupy procesów
- [N] wątki
- [T] wszystkie otwarte pliki
- [T] prywatne odwzorowania plików w pamięć

Zadanie 8 (3). Fakty dotyczące grup procesów i sterownika terminala.

- [T] Jeśli proces w grupie procesów drugoplanowych spróbuje wczytać dane z deskryptora terminala, to zostanie do niego wysłany sygnał `SIGTTIN`, który domyślnie zatrzyma proces.
- [N] Proces w grupie pierwszoplanowej oczekując na dane z terminala śpi snem nieprzerywalnym.
- [N] Po przeczytaniu znaku «`^C`» (`CTRL+C`) sterownik terminala wyśle do grup drugoplanowych sygnał «`SIGINT`».
- [T] Powłoka używa wywołania systemowego «`ioctl`», żeby ustalić bieżącą grupę pierwszoplanową.

Zadanie 9 (3). Fakty dotyczące tożsamości procesów uniksowych.

- [N] W celu sprawdzenia uprawnień dostępu do pliku jądro używa obowiązującej nazwy użytkownika i grupy.
- [N] Numery grup dodatkowych pochodzą ze zbioru grup procesów, do jakich proces należy.
- [T] Proces może wymieniać tożsamość użytkownika między dwoma numerami: «`real-uid`» i «`saved-uid`».
- [T] Proces użytkownika o `uid` > 0 może uzyskać nową tożsamość wyłącznie za pomocą wywołania «`execve`».

Zadanie 10 (3). Wartości zwracane z wywołań «`read`» i «`write`» (w tym short counts).

- [T] Wywołanie «`read`» na pliku terminala lub potoku może wczytać mniej bajtów niż zażądaliśmy.
- [T] Wywołanie «`read`» na pliku zwykłym może zwrócić mniej bajtów niż zażądaliśmy tylko wtedy, gdy kursor jest wystarczająco blisko końca pliku.
- [N] Wywołanie «`write`» na pliku zwykłym zwróci zero, gdy w systemie plików skończyło się miejsce.
- [T] Wywołanie «`write`» na gnieździe sieciowym może zapisać mniej bajtów niż zażądaliśmy, jeśli nie ma wystarczająco dużo miejsca w buforze jądra na cały pakiet.

Zadanie 11 (3). Fakty dotyczące pozostałych operacji na plikach.

- [T] Przy pomocy wywołania «`ftruncate`» można zwiększyć rozmiar pliku zwykłego.
- [N] Wywołaniem «`lseek`» można porzucić niewczytaną część bufora potoku.
- [N] Wywołaniem «`fcntl`» można zmieniać właściwości urządzeń wejścia-wyjścia reprezentowanych jako pliki.
- [T] Przy pomocy wywołania «`lseek`» można przesunąć kursor pliku poza jego koniec.

Imię i nazwisko: _____ Numer indeksu: _____

Zadanie 12 (3). Fakty dotyczące struktur danych systemu plików.

- [T] i-węzeł dużego pliku zawiera wskaźniki na strukturę przechowującą bloki należące do danego pliku.
- [N] i-węzeł zawiera nazwę pliku oraz jego typ.
- [T] Dowiązanie symboliczne mogą wskazywać na pliki, które nie istnieją.
- [N] Można tworzyć dowiązania twarde do katalogów.

Zadanie 13 (3). Fakty dotyczące potoków uniksowych.

- [T] Potoki standardu `POSIX` są jednokierunkową formą komunikacji międzyprocesowej.
- [N] Wywołanie «`read`» zwraca 0, jeśli w buforze potoku nie ma danych.
- [T] Przy zapisie bufora o długości $n \leq$ `PIPE_BUF` zakończonym sukcesem, wywołanie «`write`» zawsze zwróci n.
- [T] Gdy brak jest procesów posiadających koniec rury do odczytu, to piszący do rury dostanie sygnał «`SIGPIPE`».

Zadanie 14 (3). Fakty dotyczące gniazd.

- [T] Gniazda datagramowe umożliwiają odebranie tylko jednego pakietu przy pomocy wywołania «`recvfrom`» nawet, jeśli w buforze zmieściłyby się dwa pakiety.
- [N] Gniazda domeny uniksowej umożliwiają przesyłanie otwartych plików między procesami działającymi na dwóch różnych maszynach.
- [N] Gniazdo strumieniowe można skonfigurować jako nasłuchujące przy pomocy wywołania «`connect`».
- [N] Po zaakceptowaniu połączenia serwer komunikuje się z klientem używając gniazda nasłuchującego.

Zadanie 15 (3). Pamięć procesów uniksowych.

- [N] Wywołanie systemowe «`sbrk`» służy do zwiększania rozmiaru stosu programu.
- [T] Najmniejszą jednostką przydziału pamięci dla procesu jest jedna strona.
- [N] Poważna usterka strony powstaje wtedy, gdy proces odwołał się do adresu pod którym nie ma żadnej strony.
- [T] Wszystkie odwzorowania plików w pamięć tworzone przez wywołanie «`execve`» są prywatne.

Zadanie 16 (3). Współbieżność.

- [T] Uwięzienie (ang. *livelock*) to sytuacja, w której programy wykonują obliczenia, ale nie jest zachowana właściwość postępu.
- [N] Semafor binarny w przeciwieństwie do muteksu pamięta kto założył blokadę.
- [T] Skuteczną metodą zapobiegania zakleszczeniom jest zakładanie blokad zgodnie z ustalonym wcześniej częściowym porządkiem na blokadach.
- [N] Blokada jest sprawiedliwa, jeżeli każdy proces, który założył blokadę, zwolni ją w skończonym czasie.

Zadanie 17 (3). Wątki `POSIX`.

- [T] Każdy z wątków dostaje od systemu swój kontekst, stos i maskę sygnałów.
- [T] Wątki współdzielą przestrzeń adresową procesu do którego należą, zatem mogą modyfikować nawzajem zawartość swoich stosów.
- [T] Po zakończeniu wątku należy zwolnić zajmowaną przez niego pamięć posługując się «`pthread_join`» chyba, że wątek jest oderwany (ang. *detached*).
- [N] Algorytm szeregowania wątków działa w przestrzeni użytkownika.

Zadanie 18 (4). W każde z pustych pól wpisz dokładniej jedno niepowtarzające się polecenie (z pominięciem opcji wiersza poleceń) przy pomocy którego wyświetlisz:

Zakres adresów wraz uprawnieniami dostępu segmentów pamięci anonimowej danego procesu.	`pmap`
Numer i-węzła, właściciela i grupę, ostatni czas modyfikacji danego katalogu.	`stat`
Identyfikator procesu, który posiada nasłuchujące gniazdo sieciowe o zadanym numerze.	`netstat`
Wszystkie otwarte pliki w systemie plików zamontowanym w danym punkcie montażowym.	`lsof`

Uwaga! W poniższych zadaniach zakładamy, że wszystkie wywołania procedur wykonają się bezbłędnie, a standardowe wyjście jest niebuforowane.

Zadanie 19 (7). Plik «`two.txt`» zawiera ciąg znaków «`nilfgaard`». Przeanalizuj poniższy program i wpisz w kratkę co wydrukuje po uruchomieniu.

```
1  int main(void) {
2    char c0 = 'u', c1 = 'y', c2 = 'q';
3    char scrap[4];
4    int pid, r, r2 = open("two.txt", O_RDONLY);
5    r = dup(r2);
6    if (!(pid = fork())) {
7      read(r, &c0, 1);
8      close(r2);
9      r2 = open("two.txt", O_RDONLY);
10     read(r2, &scrap, 4);
11   } else {
12     waitpid(pid, NULL, 0);
13     read(r, &c1, 1);
14     read(r2, &c2, 1);
15   }
16   printf("%c%c%c", c0, c1, c2);
17   return 0;
18 }
```

nyquil

Zadanie 20 (7). Przeanalizuj poniższy kod i wpisz w pudełko wszystkie możliwe wydruki po uruchomieniu programu.

```
1  int main(void) {
2    int counter = 0;
3    int pid;
4    while (counter < 2 && !(pid = fork()))
5      printf(stderr, "%d", ++counter);
6    if (counter > 0)
7      printf("%d", counter);
8    if (pid) {
9      waitpid(pid, NULL, 0);
10     counter += 4;
11     printf("%d", counter);
12   }
13   return 0;
14 }
```

112254 121254 122154

Imię i nazwisko: _____ Numer indeksu: _____

UWAGA! W dalszych zadaniach zakładamy, że semafor, muteks i zmienna warunkowe są sprawiedliwe.

Zadanie 21 (8). Bariera to środek synchronizacji, o którym można myśleć jak o jednokierunkowej śluzie wodnej. Najpierw śluza otwiera pierwsze wrota i czeka na pojawienie się N statków. Po pojawieniu się N statków obsługa śluzy zamyka pierwsze wrota, obniża poziom wody i otwiera drugie wrota. Po wypłynięciu wszystkich statków obsługa zamyka drugie wrota, podwyższa poziom wody i otwiera pierwsze wrota.

Poniżej widnieje wadliwy kod operacji «wait» bariery dwuetapowej. W kratkę wpisz scenariusz, w którym implementacja nie zachowa się tak jak opisana wyżej śluza wodna.

```
1  semaphore m = 1, s1 = 0, s2 = 0
2  int n = 0
3
4  def barrier_wait():
5    wait(m)
6    n += 1
7    if n == N:
8      for i = 1 to N:
9        post(s1)
10   post(m)
11   wait(s1)
12
13   wait(m)
14   n -= 1
15   if n == 0:
16     for i = 1 to N:
17       post(s2)
18   post(m)
19   wait(s2)
```

Liczba procesów: 2N

PID	wiersze	m	s1	s2	n
1..2N	4..10	1	N	0	2N
1..N	11..18	1	0	0	N

Jednozdaniowy opis błędu:

Zakleszczenie! Warunki w liniach 7 i 15 nie zostaną już nigdy spełnione.

Zadanie 22 (6). Przeczytaj poniższy kod. Następnie określ, które z występujących w nim zmiennych są współdzielone między co najmniej dwa wątki oraz które ze zmiennych będą źródłem wyścigów w tym programie.

```
1  __thread long myid;
2  static char **strtab;
3
4  void *thread(void *vargp) {
5   myid = *(long *)vargp;
6   static int cnt = 0;
7   printf("[%ld]: %s (cnt=%d)\n",
8          myid, strtab[myid], ++cnt);
9   return NULL;
10 }
11
12 int main(int argc, char *argv[]) {
13   ...
14   strtab = argv;
15   while (argc > 0) {
16     myid = --argc;
17     pthread_create(&tid, NULL,
18                    thread, (void *)&myid);
19   }
20   ...
21 }
```

zmienna	dzielona?	wyścig?
myid	TAK	TAK
strtab	TAK	NIE
vargp	NIE	NIE
cnt	TAK	TAK
argc	NIE	NIE
argv[0]	TAK	NIE

Zadanie 23 (7). Poniżej podano częściowe rozwiązanie problemu czytelników i pisarzy, które dopuszcza głodzenie czytelników. Twoim zadaniem jest uzupełnienie implementacji procedury zakładającej i zdejmującej blokadę pisarza. Wewnątrz kratki można modyfikować zmienne «readers», «writers», «is_writing» oraz wykonywać operacje «wait», «broadcast» i «signal» na zmiennych warunkowych «can_write» i «can_read».

```
1  readers = 0
2  writers = 0
3  is_writing = False
4  mtx = mutex()
5  can_write = condvar()
6  can_read = condvar()
7
8  def rd_lock():
9    lock(mtx)
10     while writers > 0:
11       wait(can_read, mtx)
12     readers += 1
13   unlock(mtx)
14
15 def rd_unlock():
16   lock(mtx)
17     readers -= 1
18     if readers == 0:
19       signal(can_write)
20   unlock(mtx)
```

```
def wr_lock():
  lock(mtx)
    writers += 1
    while readers > 0 or is_writing:
      wait(can_write, mtx)
    is_writing = True
  unlock(mtx)

def wr_unlock():
  lock(mtx)
    is_writing = False
    writers -= 1
    if writers > 0:
      signal(can_write)
    else:
      broadcast(can_read)
  unlock(mtx)
```

Zadanie 24 (10). Mamy algorytm zarządzania pamięcią bazowany na boundary tags i dwukierunkowej liście FIFO wolnych bloków. Polityka przydziału to next-fit, gdzie pozycję kursora zaznaczono znakiem gwiazdki. Zwalniane bloki są gorliwie złączane. Pamięć jest adresowana słowami maszynowymi. Wolne bloki są oznaczone «F», a używane «U». Po dwukropku zanotowano rozmiar całego bloku włączając rozmiar nagłówka i stopki, które zajmują dwa słowa maszynowe. Każdy wolny blok jest oznaczony literą, która wyznacza jego pozycję na liście wolnych bloków zgodnie z porządkiem alfabetycznym. Wolne bloki muszą w sobie pomieścić wskaźniki na poprzednika i następnika.

Dla powyższego stanu początkowego struktur algorytmu zarządzania pamięcią wykonaj kolejno, jedno po drugim, ciąg poniższych żądań. Operacja «free» przyjmuje adres początku zawartości bloku.

`alloc(6) free(31) alloc(5) free(26) free(14)`

Końcowy stan obszaru zarządzanej pamięci (i opcjonalnie pośrednie):