Zadanie 2-01
Napiste program, v ktorom vytvorite N-procesov (N>3 a je volitelnym argumentom programu). Procesy budu postupne v stale rovnakom poradi zapisovat svoje PID do zdielanej pamati ZP (uloha serializacie). Toto ich poradie nech sa NAHODNE urci este pred ich prvym spustenim. Procesy musia obsahovat nekonecnu slucku, takze poradie zapisov sa bude cyklicky opakovat. Synchronizaciu zabezpecte semaformi. Program musi byt ukonceny po stlaceni <ctrl+c> (SIGINT) s naslednym vypisom obsahu ZP rodicovskym procesom a s uvolnenim vsetkych zdrojov. ZP implementujte ako samosynchronizujuci objekt.
Syntax: zadanie_2_01 [-h] [-n <N>]

Zadanie 2-02
Napiste program, v ktorom vytvorite N-procesov (N>3 a je volitelnym argumentom programu). Vsetky procesy najprv vypisu text a svoj PID vo formate "pred barierou: <pid>". Nasledne vypise vsetkych N procesov text "za barierou: <pid>". Ide o ulohu bariery. Procesy musia obsahovat nekonecnu slucku, takze poradie vypisov sa bude cyklicky opakovat. Poradie nesmie byt dane napevno v programe, ale bude urcene postupnym spustanim procesov. V kazdom procese pouzite mikrosleep v rozumnom intervale. Synchronizaciu procesov zabezpecte semaformi. Program musi byt ukonceny po stlaceni <ctrl+c> (SIGINT) tak, aby boli korektne uvolnene vsetky zdroje. Barieru implementujte ako samosynchronizujuci objekt.
Syntax: zadanie_2_02 [-h] [-n <N>]

Zadanie 2-03
Napiste program, v ktorom vytvorite N-procesov (N>3 a je volitelnym argumentom programu). Kazdy proces bude najprv zapisovat zvoleny <text> (zadany ako argument programu) do zdielanej pamati (ZP). Nasledne bude kazdy proces text z pamati vypisovat tak, aby text, ktory zapise i-ty proces, proces i+1 (modulo n) vypisal. Synchronizaciu procesov zabezpecte semaformi. Vypis procesov nech bezi v nekonecnej slucke, takze sa bude cyklicky opakovat (uloha serializacie). Poradie procesov nesmie byt napevno dane v programe, ale bude urcene postupnym spustanim procesov. V kazdom procese pouzite mikrosleep v rozumnom intervale. Program sa ukonci po stlaceni <ctrl+c> (SIGINT) tak, aby boli korektne uvolnene vsetky zdroje. ZP implementujte ako samosynchronizujuci objekt.
Syntax: zadanie_2_03 [-h] [-n <N>] -t <text>

Zadanie 2-04
Napiste dva programy, ktore budu striedavo zapisovat a vypisovat retazec "<pid>: <text>" do/zo zdielanej pamati tak, aby text, ktory do ZP zapise prvy proces, druhy nasledne vypisal a opacne: text, ktory zapise druhy proces, nasledne vypisal prvy (uloha randezvous). Na synchronizaciu pouzite semafory. Po zaslani signalu <ctrl-c> (SIGINT) oba procesy cyklus zapisov a vypisov ukoncia (rieste zmenou podmienky cyklu v handleri signalu) a budu pokracovat dalej za cyklom vypisom "<pid>: koniec" a korektne skoncia. Poradie spustenia programov nech neovplyvnuje spravnost fungovania programu. ZP implementujte ako samosynchronizujuci objekt.
Syntax: zadanie_2_04 [-h] -t <text>

Zadanie 2-22
Napiste program, v ktorom vytvorite N-podprocesov (N>3 a je volitelnym argumentom programu). Vsetky procesy najprv vypisu text "pred barierou" a svoj PID vo formate "pred barierou - <pid>" do zdielanej pamati. Nasledne vypise rodicovsky proces obsah zdielanej pamati a uvolni barieru. Po uvolneni bariery vypise vsetkych N procesov na monitor text "za barierou - <pid>" (uloha bariera). Procesy musia obsahovat nekonecnu slucku, takze poradie vypisov sa bude cyklicky opakovat. Poradie nesmie byt dane napevno v programe, ale bude urcene na zaciatku pred ich prvym spustenim nahodne. Synchronizaciu procesov zabezpecte semaformi. Program musi byt ukonceny po stlaceni <ctrl+c> (SIGINT) tak, aby boli korektne uvolnene vsetky zdroje. Barieru implementujte ako samosynchronizujuci objekt.
Syntax: zadanie_2_22 [-h] [-n <N>]

Zadanie 2-05

Napiste program, ktory bude komunikovat cez zasobnik (tzv. buffer – musi byt implementovany ako ADT pomocou zdielanej pamati). Zasobnik nech je samosynchronizujucim objektom!

S prepinacom -w bude program do zasobnika zapisovat <text>.

S prepinacom -r bude program zo zasobnika citat a jeho obsah zobrazi na stdout.

Program moze byt spusteny v niekolkych oknach s prepinacmi pre zapis a citanie tak, aby v danom case mal k zasobniku pristup vzdy len jeden zapisujuci program. Ale citacie programy mozu pristupovat naraz viacere k zasobniku. Pristup k zasobniku riadte semaforom. Program musi reagovat na signal SIGINT (napr. stlacenim klaves ctrl+c) korektnym skoncenim. Pre nazornost nech je text zapisovany v nekonecnej slucke a program, ktory cita, nech ma k zasobniku pristup len vtedy, ak tento nie je prazdny. Poradie spustenia roznych typov (citacich a zapisovacich) programov nech neovplyvnuje spravnost fungovania programu.

Syntax: zadanie_2_05 -h | -w <text> | -r

Priklad pouzitia (spustene v roznych oknach):

zadanie_2_05 -w text1
zadanie_2_05 -w text2
zadanie_2_05 -r
zadanie_2_05 -r

Zadanie 2-06

Napiste program, ktory bude v nekonecnom cykle SYNCHRONNE komunikovat cez frontu sprav.

S prepinacom -w bude program do fronty posielat spravu s danym typom <typ> a textom <text>.

S prepinacom -r bude program z fronty spravu s danym typom <typ> citat a zobrazovat na stdout.

Program musi byt spusteny v niekolkych oknach s prepinacmi pre odosielanie a prijimanie sprav. Synchronna komunikacia znamena, ze vysielajuci proces bude cakat na signal potvrdzujuci prijatie spravy. Program musi reagovat na signal SIGINT (napr. stlacenim klaves ctrl+c) korektnym skoncenim.

Syntax: zadanie_2_06 -h | -w <text> -t <type> | -r -t <type>

Priklad pouzitia (spustene v roznych oknach):

zadanie_2_06 -w text1 -t typ1
zadanie_2_06 -t typ2 -w text2
zadanie_2_06 -r -t typ2
zadanie_2_06 -r -t typ1

Zadanie 2-07

Napiste program, ktory bude podla svojich argumentov manipulovat s mnozinou semaforov.

Prepinac:

-c vytvori mnozinu <N> semaforov, a inicializuje ich,
-l uzamkne semafor s danym cislom <X>,
-u odomkne semafor s danym cislom <X>,
-m zmeni pristupove prava pre semafor na <mode>,
-d zrusi mnozinu semaforov.
-i zobrazi hodnoty vsetkych semaforov alebo len hodnotu <K>-teho semafora
-p zobrazi vsetky informacie zo struktury "ipc_perm"

Syntax: zadanie_2_07 -h | -c <N> | -l <X> | -u <X> | -m <mode> | -d | -i [<K>] | -p

Priklad pouzitia:

zadanie_2_07 -c 5, alebo zadanie_2_16 -l 0, alebo zadanie_2_16 -u 0, alebo zadanie_2_16 -m 660, alebo aj rozumne kombinacie: zadanie_2_16 -l 2 -u 3 -i

Zadanie 2-08
Napiste program, ktoreho detske procesy vypocitaju prvych <N> clenov Fibonaciho postupnosti komunikaciou cez kanaly nasledovne: i-ty proces (i=2..N) dostane na svoj standardny vstup (citaci koniec kanala) i-2. a i-1. clen Fibonaciho postupnosti. Vypocita i-ty. clen postupnosti a na svoj standardny vystup (zapisovaci koniec kanala) posle i-1. a i-ty. clen i+1.-mu procesu. Kazdy proces nech zobrazi na terminale retazec "<pid>: <i-ty clen>". Rodic musi najprv vytvorit vsetkych <N> procesov, az potom sa moze zacat vypocet synchronizovany pomocou semaforov! Posledny proces odovzda N-ty clen postupnosti rodicovi, a ten vypise na terminal retazec "<pid>: rodic - <N. clen>".
Syntax: zadanie_2_08 [-h] -n <N>

Zadanie 2-09
Napiste dva programy, ktore spustia dva externe shellovske prikazy, komunikujuce cez FIFO kanal. Standardny vystup prveho bude presmerovany na standardny vstup druheho a standardny vystup druheho prikazu presmerujte do suboru ako mapovanej pamati. Napiste aj treti program, ktory obsah mapovanej pamati precita a vypise. Spustane prikazy su uvedene ako argumenty na prikazovom riadku programu. Pozor, spustane prikazy mozu mat aj svoje argumenty! Mapovanu pamat implementujte ako samosynchronizujuci objekt! Zapis a citanie obsahu mapovanej pamati synchronizujte semaforom.
Syntax: zadanie_2_09_xyz [-h] <prikaz> [arg] [subor]
Priklad pouzitia:
./zad1 /bin/ps -ef && ./zad2 /bin/grep "pattern" file_memory_map && ./zad3 file_memory_map

Zadanie 2-10
Implementujte ulohu obsluhy zakaznikov cakajucich vo fronte (frontu implementujte ako samosynchronizujuci objekt implementovany zdielanou pamatou). Ak je fronta plna, zakaznik necaka a odide. V opacnom pripade sa zaradi do fronty. Zakaznik, ktory je na rade, z fronty vystupi a je obsluzeny. Zakaznikov a obsluzny personal reprezentujte procesmi. Program overte pre <N> zakaznikov (a zvolenu kapacitu fronty v argumente programu) a jeden proces obsluhy. Trvanie obsluhy zakaznika vhodne simulujte funkciou sleep a zakaznicke procesy nech opakovane ziadaju o obsluhu s oneskorenim generovanym nahodne z nejakeho rozumneho intervalu. Rodicovsky proces nech opakovane (napr. kazdu sekundu) zobrazuje informacie o pocte procesov vo fronte a informaciu, ktory proces je obsluhovany: "<num> <pid_zakaznik>". Program musi reagovat stlacenim ctrl+c (signalom SIGINT) korektnym ukoncenim.
Syntax: zadanie_2_10 [-h] <-k kapacita fronty> -z <N>

Zadanie 2-11
Napiste program, ktory vytvori detsky proces a ten bude postupne pocitat a zapisovat prvych <N> clenov Fibonaciho postupnosti do zdielanej pamati. Rodicovsky proces bude priebezne (kazdych nahodne zvolenych <msec> milisekund z rozumneho intervalu) zistovat ciastkove sucty uz zapisanych clenov v ZP a vypisovat ich. Pre kontrolu nech detsky proces po zapise vsetkych clenov vypise tiez ich sucet. Posledny sucet vypisany rodicovskym procesom sa musi zhodovat s vypisom detskeho procesu. Na synchronizaciu pouzite semafor. Format vypisu: "<pid>: <sum>". Program musi reagovat stlacenim ctrl+c (signalom SIGINT) korektnym ukoncenim. Zdielana pamat nech je implementovana ako samosynchronizujuci objekt.
Syntax: zadanie_2_11 [-h] -n <N> -t <msec>

Zadanie 2-12
Napiste program, v ktorom vytvorite <N> procesov. Kazdy proces bude mat nahodne pridruzene jednoznacne cislo (napr. z intervalu <1,N>). Proces bude mat pristup k suboru <subor> na jeho citanie, len ak je sucet cisiel asociovaných k procesom, ktore uz subor citaju, mensi ako <M>. Pristup k suboru implementujte vhodne navrhnutym samosynchronizujucim ADT. Na jeho synchronizaciu pouzite semafor. Procesy musia obsahovat nekonecnu slucku (opakovane sa budu pokusat subor citat). Program bude pri stlaceni <ctrl+c> (obdrzani signalu SIGINT) ukonceny tak, aby boli uvolnene vsetky zdroje.
Syntax: zadanie_2_12 [-h] -n <N> -m <M> -f <subor>

Zadanie 2-13
Spustite tri programy, ktore budu komunikovat zasielanim sprav do fronty sprav. Kazdy program z fronty sprav cita iba spravy urcene jemu podla specifikacie parametrom -t pri spusteni programu. V nekonecnej slucke sa vzdy pokusi precitat spravu urcenu jemu. Ak taka je, precita ju a vypise vo formate "<pid>: dostal od <pid_sender>". Nasledne nahodne zvoli, aky typ spravy posle dalej. Vypise text "<pid>: posiela <typ>", a odosle spravu. Spravou nech je pid odosielatela. Komunikaciu programov realizujte ako synchronnu (t.j. vysielajuci proces bude cakat na signal potvrdzujuci prijatie spravy). Samotnu frontu sprav (a manipulaciu s nou) implementujte ako vhodne navrhnuty samosynchronizujuci ADT! Ukoncenie programov bude prerusenim z klavesnice kombinaciou ctrl+c (alebo obdrzanim signalu SIGINT). Pritom musia byt uvolnene vsetky zdroje. Funkcnost programov overte ich spustenim v troch oknach.
Syntax: zadanie_2_13 [-h] -t <typ>

Zadanie 2-14
Napiste program, v ktorom najprv vytvorite vsetkych <N> procesov (N>3 a je volitelnym argumentom programu). Potom kazdy proces zapise text "<pid>" do suboru <subor> (ktory je mapovanou pamatou). Taktiez bude kazdy proces text z pamati citat a vypisovat tak, aby text, ktory zapise i-ty proces, proces i+1 (modulo n) vypisal vo formate: "<pid>: precital som: <pid>". Manipulaciu s mapovanou pamatou zabezpecte pomocou samosynchronizujuceho ADT. Na jeho synchronizaciu pouzite semafor. Synchronizaciu procesov zabezpecte tiez semaformi. Vypis procesov nech bezi v nekonecnej slucke, takze sa bude cyklicky opakovat (uloha serializacie). Poradie spustania procesov nech je dane hodnotami PID od najmensieho po najvacsi! Ak program obdrzi signal SIGINT (napr. z klavesnice stlacenim ctrl+c), musi korektne skoncit. Funkcnost programu nesmie byt zavisla na poradi spustania procesov!
Syntax: zadanie_2_14 [-h] [-n <N>] -f <subor>

Zadanie 2-15
Napiste program, ktory spusti tri externe shellovske prikazy, ktore budu komunikovat cez kanal. Standardny vystup prveho bude presmerovany na standardny vstup druheho a standardny vystup druheho na vstup tretieho. Ak je dany prepinac -f:

1. presmerujte standardny vystup posledneho prikazu do suboru <file>,
2. obsah suboru nasledne vypise rodicovsky proces.

Prikazy su uvedene ako argumenty na prikazovom riadku. Pozor, prikazy mozu mat svoje argumenty. Ak program obdrzi signal SIGINT (napr. z klavesnice stlacenim ctrl+c), musi korektne skoncit.
Syntax: zadanie_2_15 [-h] [-f <file>] -1 <prik1> [args] -2 <prik2> [args] -3 <prik3> [args]
Priklady pouzitia:

zadanie_2_15 /bin/ps -ef /bin/grep "pattern" /bin/wc -l
zadanie_2_15 -s /bin/ps -ef /bin/grep "pattern" /bin/wc -l file_name

Zadanie 2-16
Napiste program, ktory bude komunikovat cez mapovanu pamat pomocou suboru <file>. S prepinacom -w bude program do suboru zapisovat retazec "<pid>;<date>;<time>;<text>". S prepinacom -r bude program zo suboru citat a obsah suboru zobrazi na standardny vystup. Program moze byt spusteny v niekolkych oknach s prepinacmi pre zapis a citanie tak, aby v danom case mal k suboru pristup len jeden zapisujuci program. Ak sa jedna o citacie programy, mozu k suboru pristupovat viacere naraz. Pristup k mapovanej pamati musi byt realizovany cez vhodne navrhnuty samosynchronizujuci ADT. Synchronizaciu ADT zdielana pamat robte pomocou semaforu. Program musi reagovat na ctrl+c (alebo obdrzanie signalu SIGINT) korektnym skoncenim. Pre nazornost nech je text zapisovany v nekonecnej slucke a program, ktory cita, nech ma k suboru pristup, len ak tento nie je prazdny.
Syntax: zadanie_2_16 [-h] -w <file> -t <text> | -r <file>

Zadanie 2-17
Napiste program, v ktorom vytvorite <N> procesov (N>3 a je volitelne). Procesy budu postupne cyklicky zapisovat svoje PID do suboru <file> ako mapovanej pamati (uloha serializacie). Pristup k mapovanej pamati musi byt realizovany pomocou vhodne zvoleneho samosynchronizujuceho ADT. Synchronizaciu ADT robte pomocou semaforu. Procesy musia obsahovat nekonecnu slucku, takze poradie zapisov sa bude opakovat. Ich synchronizaciu zabezpecte semaformi. Poradie spustania procesov nech je dane nahodne este pred ich prvym spustenim. Program musi byt ukonceny po stlaceni <ctrl+c> (obdrzani signalu SIGINT) s naslednym vypisom obsahu mapovanej pamati rodicovskym procesom a s korektnym uvolnenim vsetkych zdrojov.
Syntax: zadanie_2_17 [-h] [-n <N>] -f <file>

Zadanie 2-18
Napiste programy typu vyrobca a konzument, ktore budu vkladat a vyberat zo zasobnika ako mapovanej pamati (danej suborom <file>) nizsie specifikovane udaje. Vkladanie a vyber nemoze byt sucasny. Na vzajomne vylucenie procesov pouzite vhodne zvoleny samosynchronizujuci ADT mapovana pamat. Nech sa synchronizuje pomocou semaforu. Pri preteceni alebo podteceni zasobnika musia procesy cakat. Kapacita zasobnika je urcena argumentom -c. Vyrobcovia budu v nekonecnej slucke zapisovat do zasobnika nahodne realne cisla v rozsahu 1 az 10. Kazdy vyrobca pri ukonceni svojho behu vypise sucet cisiel, ktore zapisal, vo formate "<pid>: vyrobca <sum>". Pokym je zasobnik neprazdny, konzument v jednom cykle slucky vyberie prave jedno cislo zo zasobnika, a odovzda riadenie procesoru. Ak konzument zisti, ze zasobnik je prazdny, ukonci svoj beh a vypise sucet precitanych cisiel vo formate "<pid>: konzument <sum>". Za predpokladu, ze na zaciatku a na konci je zasobnik prazdny, suma, ktoru vypisu vyrobcovia, sa musi rovnat sume, ktoru vypisu konzumenti. Programy vyrobcu musia reagovat na stlacenie ctrl+c (alebo zaslanie signalu SIGINT) korektnym skoncenim (a vypisanim suctu zapisanych cisiel). Funkcnost demonstrujte aspon na 4 vyrobcoch a 4 konzumentoch.
Syntax: zadanie_2_18 [-h] -f <file> -c <cap>

Zadanie 2-19

Napiste programy typu vyrobca a konzument, ktore budu vkladat a vyberat zo zasobnika (ktory je implementovany zdielanou pamatou) nizsie specifikovane udaje. Vkladanie a vyber nemoze byt sucasny. Na vzajomne vylucenie procesov pouzite vhodne zvoleny samosynchronizujuci ADT zdielana pamat. Nech sa synchronizuje pomocou semaforu. Pri preteceni alebo podteceni zasobnika musia procesy cakat. Kapacita zasobnika je urcena argumentom -c. Vyrobcovia budu v nekonecnej slucke zapisovat do zasobnika nahodne realne cisla v rozsahu 1 az 10. Kazdy vyrobca pri ukonceni svojho behu vypise sucet cisiel, ktore zapisal, vo formate "<pid>: vyrobca <sum>". Pokym je zasobnik neprazdny, konzument v jednom cykle slucky vyberie prave jedno cislo zo zasobnika, a odovzda riadenie procesoru. Ak konzument zisti, ze zasobnik je prazdny, ukonci svoj beh a vypise sucet precitanych cisiel vo formate "<pid>: konzument <sum>". Za predpokladu, ze na zaciatku a na konci je zasobnik prazdny, suma, ktoru vypisu vyrobcovia, sa musi rovnat sume, ktoru vypisu konzumenti. Programy vyrobcu musia reagovat na stlacenie ctrl+c (alebo zaslanie signalu SIGINT) korektnym skoncenim (a vypisanim suctu zapisanych cisiel). Funkcnost demonstrujte aspon na 4 vyrobcoch a 4 konzumentoch.
Syntax: zadanie_2_19 [-h] -c <cap>

Zadanie 2-20

Napiste klientsky program pre aplikaciu typu klient/server. Aplikacia sluzi na komunikaciu cez schranku <socket> v tom istom uzle. Pre cast servera sluzi vypracovanie zadania c. 21. Pre komunikaciu pouzite dialog:

Klient zasle spravu <typ1>, napr: "Som pripojeny. Pocujes ma?"
Server obdrzi spravu <typ1> a odpovie spravou <typ2>, napr: "Ja ano a ty?"
Klient obdrzi spravu <typ2> a odpovie spravou <typ3>, napr: "Ja ta tiez pocujem."

Po jednom behu cyklu sa klient na nahodne zvoleny cas (v rozumnom intervale do 57 sekund) odmlci, a znovu opakuje komunikaciu. Ukoncenie programu bude pomocou prerusenia z klavesnice ctrl+c (alebo zaslanim signalu SIGINT), pricom musia byt uvolnene vsetky zdroje. Pri obdrzani signalu na ukoncenie musi klient korektne ukoncit spojenie so serverom.
Syntax: zadanie_2_20 [-h] -s <socket>

Zadanie 2-21

Napiste program servera pre aplikaciu typu klient/server. Aplikacia sluzi na komunikaciu cez schranku <socket> v tom istom uzle. Pre cast klienta sluzi vypracovanie zadania c. 20. Pre komunikaciu pouzite dialog:

Klient zasle spravu <typ1>, napr: "Som pripojeny. Pocujes ma?"
Server obdrzi spravu <typ1> a odpovie spravou <typ2>, napr: "Ja ano a ty?"
Klient obdrzi spravu <typ2> a odpovie spravou <typ3>, napr: "Ja ta tiez pocujem."

Server musi dokazat obsluhovat minimalne 23 klientov. Ak sa klient do 81 sekund znovu neohlasi, server korektne ukonci jeho obsluhu. Ukoncenie programu bude pomocou prerusenia z klavesnice ctrl+c (alebo zaslanim signalu SIGINT), pricom musia byt uvolnene vsetky zdroje. Takisto pri ukonceni musi server korektne ukoncit vsetky aktivne spojenia. Pri prijati alebo zruseni spojenia o tom server musi podat informaciu v rozumnom tvare.
Syntax: zadanie_2_21 [-h] -s <socket>