Klasa Thread
Name – Thread.CurrentThread.Name, Priority – enum {Lowest, Normal, Highest}, IsAlive, ThreadSafe, IsBackground
Start(), Abort(), Suspend(), Resume(), Join(), Sleep()

Foreground i background niti – foreground po defaultu. Odrzavaju app u zivotu dok se bar 1 izvrsava. Nakon zavrsetka svih foreground niti zavrsava se i program zajedno sa svim background nitima, preskacu se svi finally blockovi. Niti iz thread poola su background threads

Stanja niti – Aborted, AbortRequested, Background, Running, Stopped, StopRequested, Suspend, SuspendRequested, Unstarted, WaitSleepJoin

Thread.Interrupt – odblokiranje niti koja ceka ulaz u neku kriticnu sekciju. Generise se ThreadInterruptException, ako nit nije blokirana nista se ne desava
Thread.Abort – ocigledno

Thread.SpinWait() (interno je koriste monitori i readerwriterlock?) - nit se ne blokira ali odredjeni broj iteracija ne radi nista
Thread.Sleep() - spava odredjeno vreme ili do poziva thread.interrupt(), Thread.Sleep(0) – dodeliti procesor drugoj niti

ThreadSafe = nema problema sa izvrsavanjem koda u visenitnom okruzenju

Zakljucavanje – samo jedna nit moze da udje u kriticnu sekciju koda. Druge niti se smestaju u fifo red. Objekti koji mogu da se koriste za zakljucavanje su samo referentni tipovi podataka.

Lock – sintax sugar za Monitor.Enter i Monitor.Exit + try catch blokovi. Nit koja drzi lock moze ponovo da ga pribavi i oslobodi. Ostale niti cekaju da se lock oslobodi. Mozemo da nestujemo lockove. Oslobadjanje najugnjezdenijeg locka ne znaci da druga nit moze da ga pribavi. Blokiranje niti je skupa operacija.

WaitHandle – omogucava uzajamno iskljucivanje i omogucava sinhronizaciju
EventWaitHandle – komunikacija izmedju niti koriscenjem signala. Jedna ili veci broj niti su blokirane na objektu sve dok neka druga nit ne posalje signal (poziv metode Set). Postoje 2 moda AutoReset i ManualReset. Kod AutoReseta kada odblokiraju nit automatski se vracaju u nesignalizirano stanje. ManualReset odblokira sve niti i ostaje u signaliziranom stanju sve dok se ne pozove metoda Reset. Nad WaitHandleom moze da se poziva
WaitOne – blokira nit dok taj objekat ne postane signaliziran
WaitAll – blokira nit dok sve niti ne pozovu set
WaitAny - ...

Mutex – mogu da se koriste i za sinhronizaciju procesa. Sporiji od locka puno. Koristi se WaitOne i ReleaseMutex. Nit koja pribavi muteks mora i da ga oslobodi.

Semafor – semafor moze da zakljuca i otkljuca bilo ko. SemafoSlim brza varijanta

SinhronizacioniKontekst – automatsko medjusobno iskljucivanje niti. Klasa se izvede iz ContextBoundObject – samo jedna nit u jednom trenutku moze da izvrsava fje ove klase

Klasa Interlocked omogucava da se jednostavne operacije mogu obavljati nad deljivim resursom bez potrebe za zakljucavanjem ili blokiranjem. Interlocked.Add, Interlocked.Increment
Thread.VolatileRead/Write – citanje upis direktno iz memorije

- Barijera
-Monitor

ReaderWriterLock – implementacija problema citaoci pisci. Dozvoljava vecem broju niti da pristupa radi citanja, ali samo jednoj radi upisa. AcquireReaderLock, AcquireWriterLock, Release...Lock. Postoje 2 reda cekanja, nit iz write reda ce da bude pustena tek kad se ovaj drugi isprazni. Slim verzija: Enter/Exit ... Lock. Dozvoljava rekurziju (ista nit rekurzivno zahteva lock i brojack locka se povecava)
Moguca je konverzija lockova: UpgradeToWriterLock DowngradeFromWriterLock

ThreadPool – obezbedjuje odredjen broj background niti (worker threads) kojima sistem upravlja. Za svaki proces se kreira samo 1 thread pool, a broj raspolozivih niti se moze menjati pomocu SetMaxThreads. Po zavrsetku posla niti se ne zavrsavaju vec se ponovo koriste. Thread pool eliminise overhead koji postoji prilikom kreiranja dodatnih niti.
Nad tp moze da se pozove
RegisterWaitForSingleObject – koristi se u slucajevima kada app ima veliki broj niti koje najveci deo vremena provode u stanju cekanja. Ova metoda prihvata delegate koji se izvrsavaju kada wait handle bude signaliziran. Dok nije signaliziran ne zauzima se nit.
QueueUserWorkItem – pozivom ove metode odmah se krece u obradu koriscenjem niti iz thread poola, ukoliko postoji slobodnih niti. U suprotnom funkcija se smesta u red cekanja

BackgroundWorker – klasa za rad sa worker nitima. Omogucava otkazivanje niti, postoji protokol za pracenje progresa i koristi niti iz threadpoola.

AsinhroneDelegateFunkcije omogucavaju da se komunikacija sa nitima iz threadpoola odigrava u oba smera – da se posalju neke vrednosti kao parametri ali i da se vrate vrednosti iz niti.

Timer – postoji 4 vrste tajmera: 2 singlethreaded di 2 multithreaded

ThreadLocalStorage – LocalDataStoreSlot = Thread.GetNamedDataSlot()

KONKURENTNO PROGRAMIRANJE
Izvrsavanje nekoliko programa ili delova programa u isto vreme. Konkurentnost se moze obezbediti kroz multitasking ili kroz asinhrono izvrsavanje zadataka
Kod CPU sa jednim jezgrom, konkurentno izvrsavanje ne podrazumeva da se razliciti programi ili delovi programa izvrsavaju simultano. Kod ovakvih sistema koristi se time slicing.
Paralelno programiranje podrazumeva konkurentno izvrsavanje veceg broja taskova istovremeno. Paralelizam zahteva odgovarajucu hardversku podrsku, odnosno veci broj cpu. Paralelni program je istovremeno i konkurentni program. Obrnuto ne mora da vazi.

Visenitno programiranje multithreading je prosirenje koncepta multitaskinga ciji je cilj da unapredi performanse programa. Visenitno programiranje podrazumeva konkurentno izvrsavanje, ali konkurentno programiranje ne mora da podrazumeva visenitnost.

Problemi kod konkuretnog programiranja: nepredvidljivost, ne moze se postici jednostavnim pokretanjem veceg broja niti.

Proces predstavlja osnovnu jedinicu za dodelu resursa od strane os-a (pod resurs se podrazumeva mem, soketi, ui kanali i sl). Procesi su medjusobno nezavisni i izolovani, ne dele resurse.

Model konkurentnog programiranja zasnovan na koriscenju procesa nije najbolje resenje: kreiranje i unistavanje procesa su skupe operacije. U multitasking rezimu cena context switcha je velika (azuriranje tlb-a, i tabela za prevodjenje adresa). Komunikacija zahteva posredstvo os-a.

Kod novih os proces se sastoji od veceg broja konkurentnih niti izvrsenja. Nit predstavlja osnovnu jedinicu izvrsenja kod os-a. Svaka nit koja pripada istom procesu dele resurse tog procesa. Nit poseduje stek, context registre i thread local storage.
Prednosti koriscenja niti: manje vremena je potrebno da se kreira nova nit. Kod windowsa su procesi skupi i niti jeftine. Manje zahtevne u pogledu resursa, brza komunikacija i promena konteksta. Greska u okviru jedne niti srusice sve druge niti.

Tipovi niti:
KLT – niti su direktno podrzane od strane os-a. Jezgro obezbedjuje kompletan interfejs za rad sa nitima. Jezgro je zaduzeno za rasporedjivanje niti, moguce je eksploatisati veci broj procesora. Zahteva se context switching. Blokiranje jedne niti ne blokira sve druge.

ULT – niti na korisnickom nivou. Jezgro nije svesno postojanja niti. Nitima upravlja program koji se izvrsava u korisnickom rezimu i upravljanje se zasniva na specijalizovanim bibliotekama. Promena niti je vrlo jednostavna.Blokiranje niti blokira ceo proces (jacketing), ne moze se eksploatisati veci broj procesora.

Modeli niti:
Many to one (green threads): vise niti sa korisnickog nivoa je mapirano na jednu nit na nivou os-a.
One to one: klt, najjednostavniji model, svaka nit na korisnickom nivou odgovara jednoj niti na nivou jezgra.
Many to many: hibridni model. Biblioteka je zaduzena za rasporedjivanje niti na nivou korisnika. Promena konteksta je brza, obavlja se bez os-a, ali se zahteva dobra koordinacija izmedju planera na korisnickom i kernel nivou.

Fibers – specijalan tip izvrsenja niti koji podrazumeva kooperativno rasporedjivanje (rasporedjivanje bez prekidanaja). Fiber nit koja se izvrsava mora eksplicitno da preda kontrolu drugoj niti. Thread safety nije problem, ali se ne moze efikasno iskoristiti vise jezgara.

Light weight procesi – linux niti, erlang procesi. Virtualne niti (=many to many)

Problem trke: postojanje deljivih promenljivih moze da dovede do problema trke. Nastake kada vise niti pokusava da pristupi podatku pri cemu bar jedna od njih pokusava da upise neku vrednost. Postoje read-modify-write izazvani problemi i check-than-act problemi.

Kriticna sekcija je deo koda u kome se pristupa deljivoj promenljivoj. Osnovna ideja je da se dozvoli da samo jedna nit istovremeno pristupa deljivom resursu. Uzajamno iskljucivanje.
Pravila: nit ostaje u kriticnoj sekciji samo odredjeno vreme. Nit koja zahteva ulaz u kriticnu sekciju ne moze beskonacno dugo biti zaustavljena. Jedna od tehnika koja se koristi je zakljucavanje kriticne sekcije.
Zakljucavanje kriticne sekcije se moze postici: Zabranom prekida. Tipicno ulazak niti u kriticnu sekciju sprecava njeno prebacicvanje na drugi cpu sve dok ona ne napusti kriticnu sekciju, Planer koji treba da prekine nit koja je u ks ce ili dozvoliti niti da zavrsi ks ili dodeliti jos jedan kvant. Ks se izvrsava na istom proc od trenutka kada nit udje dok ne izadje iz ks. Tako je izbegnuta sinhro izmedju jezgara.

Za program se kaze da je thread safe ako je njegovo izvresenje u visenitnom okruzenju predivdljivo i izbegavaju se nezeljeni efekti. Kod je ts ili nije ts. Postoje 2 pristupa za izbegavanje problema trke prilikom implementacije thread safety-ja:
Izbegavanje deljivih resursa ili koriscenje sinhronizacije.
Izbegavanje: re-entrancy – kod je organizovan tako da niti ne koriste globalne i staticke promenljive. Nakon sto su prekinute, nastavljaju odakle su stale. Pristup podacima koji su deljivi samo atomicnim operacijama.
Thread local storage – promenljive su lokalizovane tako da svaka nit ima kopiju promenljivih.
Immutable objects

Sinhronizacija: medjusobno iskljucenje (pristup podacima serijalizovan tako da samo jedna nit moze da pristupi deljivom podatku) ili atomicne operacije (ne mogu biti prekinute od strane drugih niti, zahteva se koriscenje spec masinskih instrukcija).

Primitivne atomicne operacije: testandset, compareandswap, atomic
Busy waiting, spining – nit u petlji proverava i ceka da neki uslov bude ispunjen.
Spin lock – kada nit zahteva spin lock koji je vec zauzet pokrece petlju u ukojoj ispituje da li je lock postao dostupan. Nema blokiranja niti i troskova oko blokiranja/pokretanja niti – context switch, ali nit koristi procesor. Ima smisla koristiti u situacijama kada je nophodno obezbediti kratkotrajno blokiranje niti.

Semafor – celobrojna nenegativna promenljiva s nad kojom su definisane dve atomicne operacije: wait ili P, signal ili V. Za svaki semafor vezan je red cekanja u koji se smestaju niti nakon blokiranja. Prilikom izvrsenja operacije V i odblokiranja niti, nije definisano okja nit ce biti odblokirana. Najcesce se koristi fifo impl. Ideja je da se kriticna sekcija uokviri pozivima p i v operacija. Semafor koji dozvoljava da brojacka promenljiva ima vrednost != od 0 i 1 je brojacki semafor. Koristi se za kontrolu pristupa skupu resursa. Moguci problemi: nit zahteva resurs i ne oslobadja ga, nit oslobodi resurs koji nije zauzela, nit drzi resurs dugo iako joj ne treba, nit koristi resurs a nije ga zahtevala.
P i V moraju da butu atomicne. Za implementaciju koriste se masinske instrukcije testandset compareandswap i sl. U odsustvu moze da se koristi busy waiting, dekerov algo, petersonov i sl.

Binarni sem se zovu i muteksi iako ima razlike. Muteks moze da oslobodi samo nit koja ga je zauzela. Ovim se resavaju neki problemi koji postoje kod semafora. Nit koja drzi muteks ne moze biti prevremeno zavrsena, nit koja trenutno drzi muteks ima najveci prioritet. Dedlok pri zavrsavanju niti (os moze da joj oduzme vlasnistvo nad muteksom)

Uslovne promenljive – mehanizam za sinhronizaciju niti. Uslovne promenljive se koriste kada nit treba da utvrdi da li je uslov ispunjen ili ne kako bi nastavila sa izvrsenjem. Alternativa je busy wait sa cekanjem da se uslov ispuni. Za usl prom je vezan red cekanja. Kada uslov nije ispunjen nit se blokira i smesta u taj red cekanja. Kada druga nit ispuni uslov salje signal koji odblokira jednu od niti. Ako nema blokiranih niti signal se gubi.
Definisane su tri operacije wait c, m – c usl promenljiva, m muteks, signal c i braodcast c

Monitor – konstrukcija programskog jezika, objekat koji omogucava medjusobno iskljucenje niti ali i mogucnost da cekaju do ispunjenja odredjenog uslova. Svakoj metodi monitora moze pristupiti samo jedna nit. Monitor mora da obezbedi mehanizam da se niti privremeno odreknu ekskluzivnog pristupa deljivom resursu dok cekaju da odredjeni uslov bude ispunjen. Obicno se sastoji od muteksa i uslovnih promenljivih.
Postoje 2 vrste monitora Hoare i MESA. Kod Hoare monitora imamo blokirajuce uslovne promenljive. Nit koja poziva operaciju signal odmah napusta monitor, a izvrsavanje u monitoru nastavlja neka druga nit. Kod MESA monitora nit koja je pozvala signal i dalje nastavlja da se

izvrsava u monitoru. Nakon poziva signal neka od niti se iz reda cekanja smesta u red spremnih niti. Prednosti monitora: laksi za koriscenje od semafora, izbegavaju se potencijalni problemi. Ali predstavlja sinatksni secer. Kompajler je zaduzen da generise odgovarajuci kod oko monitora.